UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

FACULTAD DE CIENCIAS

Algunas heurísticas aplicadas al Problema del Agente Viajero

T E S I S

QUE PARA OBTENER EL TÍTULO DE:

ACTUARIA

P R E S E N T A :

ALLISON ODETTE MERINO TREJO

TUTORA:

DRA. en I.O CLAUDIA ORQUÍDEA LÓPEZ SOTO

CD. MX. 2023

A mis padres
Victoria y Leopoldo

Agradecimientos

Agradezco todo el tiempo, paciencia, enseñanza, las oportunidades académicas, confianza y valiosos consejos que me ha brindado mi tutora, Claudia Orquídea.

Agradezco a David Chaffrey por darme la oportunidad de dar mi primera ayudantía, por sus buenos consejos y pláticas, por su amistad y confianza que me ha tenido en estos años.

También agradezco el tiempo que se tomaron mis sinodales, Javier Ramírez, Zaida Estefanía, Adrián Girard y Ana Lilia para revisar y brindarme sus valiosas observaciones de mi trabajo.

Agradezco a Luis Vicente por darme todo su amor, cariño, paciencia, bonitos recuerdos y por ser un gran compañero de vida en estos años.

Agradezco a mi mejor amigo de toda la carrera Carlos Fernando quien hizo que todas las clases fueran divertidas y por su incondicional apoyo.

Y finalmente, agradezco a mi familia por su apoyo durante mi formación académica, a mis papás Victoria y Leopoldo por darme todo su amor, comprensión, tiempo y por dejarme vivir cada etapa como lo he planeado. A mi hermana Yery por quererme, enseñarme, cuidarme y guiarme con sus valiosos consejos, y a mi hermana Frida por siempre apoyarme en todo momento, aún en las desveladas. Y también a Otto Ulrike por ayudarme hacer mis tareas de la plataforma de inglés.

Índice general

Introducción

¡eureka, eureka!
Arquímedes

Siempre es interesante leer el logro de los científicos a través del tiempo. Gracias a la contribución de matemáticos y físicos se han desarrollado grandes proyectos importantes en la historia del ser humano, como lo hizo Arquímedes con su famoso descubrimiento conocido como *el principio de Arquímedes* el cual dice que todo cuerpo sumergido en un fluido recibe de éste un empuje vertical y hacia arriba igual al peso del fluido que desaloja. La mención de Arquímedes es para decir que su famosa exclamación «¡heurẽka, heurẽka!» da origen a la palabra *heurística* y es el nombre que reciben los métodos utilizados para trabajar con el problema del Agente Viajero.

Para el presente trabajo se eligió el problema matemático conocido como el *Problema del Agente Viajero*, el cual tiene un gran número de aplicaciones importantes en distintas áreas como por ejemplo: el problema del Ruteo de Vehículos tiene como objetivo minimizar los costos de transporte vinculados a rutas de reparto. En la historia del problema se dieron cuenta que no era tan fácil de resolverlo y que un algoritmo simple no garantizaba una buena solución por lo que algunos matemáticos se dedicaron a estudiarlo y gracias a ellos ahora se tienen un gran número de técnicas y herramientas que han sido utilizadas para continuar trabajándolo.

El objetivo del presente trabajo es implementar, comparar y analizar los resultados de las heurísticas elegidas: recocido simulado y algoritmos genéticos aplicadas al Problema del Agente Viajero.

El trabajo está compuesto de la siguiente manera:

Capítulo 1. En este capítulo se presenta lo más reciente de la historia del problema del agente viajero y se muestra su planteamiento matemático. Se explican conceptos básicos de gráficas con el fin de comprender el planteamiento del Problema del Agente viajero.

Capítulo 2. El capítulo comienza con una breve introducción a la complejidad para hacer énfasis en la importancia de las heurísticas asociadas a un problema. Posteriormente, se muestra la teoría inicial correspondiente a los conceptos de *solución vecina, construcción de vecindades, algoritmos de optimización* y finalmente se presenta la definición de *heurística*.

Capítulo 3. Se explica el concepto de *búsqueda local* que es la base de muchos métodos heurísticos por lo que juega un papel importante. También se explica la construcción de vecindad usada en este trabajo para el problema del agente viajero, adicionalmente se define la manera de construir una solución inicial.

Capítulo 4. Se presenta la primera heurística conocida como *recocido simulado*. Se inicia con la historia sobre su origen y después se explica la manera general de trabajarlo para cualquier problema de optimización. Después se explica la adaptación de los parámetros correspondientes al problema del agente viajero y se describe un ejemplo.

Capítulo 5. En este capítulo se explica la segunda heurística usada que es *algoritmos genéticos*. Se habla brevemente de los antecedentes de la heurística así como de la relación entre la terminología biológica y la adaptación al problema del agente viajero, y algunos operadores y criterios utilizados por la heurística.

Capítulo 6. Finalmente, se encuentra el capítulo donde se explica la implementación de las heurísticas abordadas para el problema del agente viajero. Se mencionan las instancias con las que se trabajó y el sitio web donde se recuperaron los datos. Y se dan las tablas de comparación de resultados dando como resultado la conclusión del trabajo.

Capítulo 7. En este capítulo se concluye si el objetivo del trabajo se cumplió, también se habla de lo que se espera después de haber realizado las implementaciones elegidas y se menciona la heurística que tuvo un mejor rendimiento.

Capítulo 1

El Problema del Agente Viajero

1.1. El Problema del Agente Viajero y sus antecedentes

El estudio del Problema del Agente Viajero (*The traveling salesman problem*) comenzó en el siglo dieciocho por los matemáticos William Rowam Hamilton y Thomas Kirkman. En 1930, Karl Menger fue el primero que estudió el planteamiento general del problema en Viena y Harvard. Más adelante, el problema fue promovido por Hassles, Whitney y Merril en Princeton [8].

El famoso problema del agente viajero fue planteado al pensar en un vendedor o agente de negocios que necesita promover y vender su producto, quien emprende su recorrido desde la ciudad donde reside, llamándole a ésta la ciudad inicial, y debe recorrer una lista de n ciudades representadas por puntos en un mapa cumpliendo la condición de visitar cada ciudad una sola vez y al final regresar a la ciudad inicial. Las distancias entre cada una de las ciudades de la lista pueden ser calculadas, ya que se conocen las coordenadas de los puntos. Partiendo de la premisa anterior surge la siguiente pregunta, ¿cuál es el tour de menor distancia que cumpla las

condiciones del problema del agente? Fue así que nació el famoso problema del agente viajero (PAV).

Hay distintas aplicaciones para el PAV que han sido planteadas en algunas páginas web donde se tienen distintos objetivos, por ejemplo: dada una lista de coordenadas de ciudades se busca formar un rompecabezas de la Mona Lisa; dadas las coordenadas de algunas estrellas se busca realizar el mapa tridimensional más grande y preciso de la galaxia; entre otros problemas planteados por distintos investigadores. [1]

Una de las aplicaciones modernas es aplicar el modelo del PAV al juego de Pokémon Go en donde el jugador desea encontrar la ruta más corta dado que tiene la lista de coordenadas de las pokeparadas de un cierto lugar. Es decir, si se tiene la lista de las pokeparadas, estas pueden verse como ciudades y al final el objetivo del problema es recorrerlas todas con la condición de que se visiten una sola vez utilizando la menor distancia posible. Para ilustrar el ejemplo en la figura 1.1 se encuentra un mapa que representa un conjunto de 99 pokeparadas en la ciudad de San Francisco, Estados Unidos. El mejor tour que se encuentra tiene una distancia total de 104,503 metros.

Entonces, si el jugador es curioso y sabe algo de optimización pensará en resolver el problema utilizando algún algoritmo que le permita obtener la mejor solución para recorrer estas pokeparadas con la menor distancia. Una vez que comience a ejecutar su código y vea el tiempo empleado podrá darse cuenta de que lo mejor es utilizar estrategias eficientes como las heurísticas, las cuales serán definidas más adelante.

[1] Pueden consultarse estos ejemplos en la siguiente página: http://www.math.uwaterloo.ca/tsp/data/

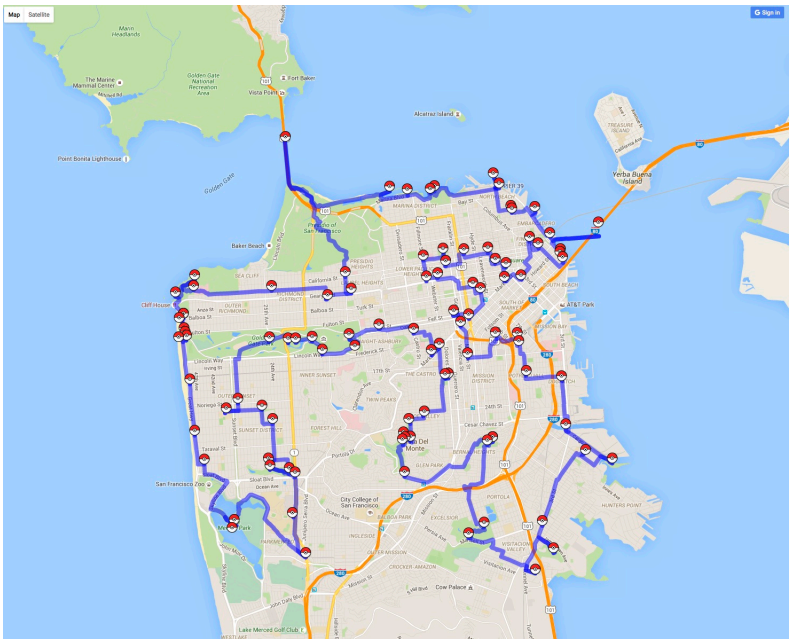


Figura 1.1: Ejemplo del mapa con 99 pokeparadas en San Francisco, EE.UU. https://www.math.uwaterloo.ca/tsp/poke/index.html

1.2. Algunos conceptos básicos de gráficas

Para entender el planteamiento y la explicación matemática de los siguientes capítulos se resumen algunas definiciones y conceptos básicos de gráficas [3].

Una gráfica es una pareja de puntos y líneas denotada como $[X, A]$, donde X es un conjunto de puntos llamados *vértices* o *nodos* y A es un conjunto de líneas, llamados arcos o aristas que unen todos o algunos de los vértices. La notación es $G = [X, A]$.

Se dice que una gráfica G es *dirigida* u orientada, si los elementos de A definen una dirección, a estos elementos se les llaman arcos. De lo contrario diremos que la gráfica G es *no dirigida* y a los elementos de A se les llaman aristas.

Un arco puede representarse como la pareja (i, j), donde los elementos $i, j \in X$ son los vértices que unen dicho arco. Denotamos el arco $a = (i, j) \in A$ con i el vértice o extremo inicial de a y j el vértice o extremo final de a. En el caso de una arista, se denota de la misma manera $a = (i, j) \in A$ pero como no hay sentido, se dice que i es el vértice o extremo inicial de a y j es el vértice o extremo final de a o viceversa.

Un arco de la forma (i, i), es llamado bucle o rizo. Una gráfica *simple* es aquella que no tiene bucles y además un arco es el único que une dos vértices cualesquiera.

Sea G una gráfica dirigida, se define el grado exterior de un vértice i, como el número de arcos que lo tienen como extremo inicial, se denota $g^+(i)$. El grado interior de un vértice i es el número de arcos que lo tienen como extremo final, se denota $g^-(i)$. Para una gráfica con m arcos y n vértices, se cumple que la suma de los grados exteriores de los n vértices es igual a la suma de los grados interiores de estos mismos y a su vez es igual al número de arcos, lo cual está representado en la ecuación 1.1.

$$\sum_{i=1}^{n} g^+(i) = \sum_{i=1}^{n} g^-(i) = m \tag{1.1}$$

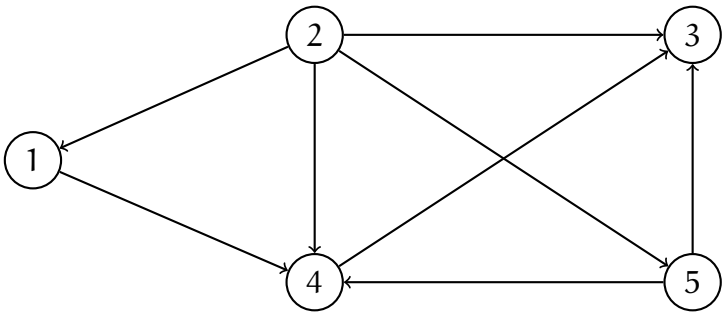


Figura 1.2: Gráfica de $m = 8$ arcos y $n = 5$ vértices.

De la figura 1.2 se tiene que:

$$\begin{aligned}
\sum_{i=1}^{5} g^+(i) &= g^+(1) + g^+(2) + g^+(3) + g^+(4) + g^+(5) \\
&= 1 + 4 + 0 + 1 + 2 \\
&= 8 \\
\sum_{i=1}^{5} g^-(i) &= g^-(1) + g^-(2) + g^-(3) + g^-(4) + g^-(5) \\
&= 1 + 0 + 3 + 3 + 1 \\
&= 8
\end{aligned}$$

Por lo tanto,

$$\sum_{i=1}^{5} g^{+}(i) = \sum_{i=1}^{5} g^{-}(i) = m = 8$$

Un vértice j es vecino adyacente de un vértice i si existe una arista $(i,j) \in A$ que los una. La siguiente función $\Gamma : X \to \wp(X)$ define los vecinos de i en la ecuación 1.2:

$$\Gamma(i) = \{j \in X | (i,j) \in A \quad o \quad (j,i) \in A\} \tag{1.2}$$

Por ejemplo, en la figura 1.2 el número de vecinos para el vértice 5 son: $\Gamma(5) = \{(5,4),(5,3),(2,5)\}$.

El *grado* de $i \in X$ es el número de arcos que tienen a i como extremo, se denota como $g(i)$ y puede verse como el número de vecinos de i, es decir, $g(i)$ es la cardinalidad de $\Gamma(i)$.

Sea G una gráfica, n el número de vértices y m el número de arcos, se cumple que la suma de los grados de los n vértices en G es igual al doble del número de arcos, lo cual se representa en la ecuación 1.3. Y se cumple para gráficas dirigidas y no dirigidas.

$$\sum_{i=1}^{n} g(i) = 2m \tag{1.3}$$

Además se cumple la siguiente igualdad $g(i) = g^{+}(i) + g^{-}(i)$, que relaciona el grado exterior con el grado interior para cada vértice.

Un *camino* o *trayectoria positiva* está conformado por una secuencia de arcos $a_1, a_2, ..., a_q$ en donde el nodo final a_i es el nodo inicial de a_{i+1}.

Un *camino simple* es un camino $i_1, a_1, i_2, a_2, ..., a_{q-1}, i_q$ tal que $a_i \neq a_j$ si $i \neq j$, es decir, los arcos involucrados son distintos.

Un *camino elemental* es un camino $i_1, a_1, i_2, a_2, ..., a_{q-1}, i_q$ tal que $i_j \neq i_k$ para $j \neq k$, es decir, los nodos de la secuencia son distintos. Y un camino elemental es también simple.

Dados dos vértices $i_j, i_k \in X$ de G, se dice que están *conectados* si existe un camino de i_j a i_k o viceversa.

Un *circuito* es un camino $i_1, a_1, i_2, a_2, ..., a_{q-1}, i_q$ donde el nodo final de a_{q-1} y el inicial de

a_1 coinciden, entonces, un circuito es un camino "cerrado".

Una *cadena* o *trayectoria* es una secuencia de aristas (o arcos) $a_1, a_2, \cdots a_q$ donde toda arista está conectada con otra. Se puede expresar como una secuencia de nodos y arcos o aristas, para el caso de una gráfica dirigida la secuencia de nodos puede ser poco descriptiva, ya que existe la posibilidad de que entre dos nodos haya dos o más arcos que los una, entonces para estas gráficas puede utilizarse la siguiente notación: $i_1 \rightarrow i_2 \leftarrow \cdots \rightarrow i_q$, donde las flechas indican el sentido del arco entre cada par de nodos o bien se puede denotar de la siguiente manera: $C : i_1 \rightarrow i_q$, en la cual se indica que la cadena tiene extremo inicial al nodo i_1 y extremo final al nodo i_q.

De manera análoga se tienen las definiciones de cadena simple y cadena elemental.

Un *ciclo* es una cadena $i_1, a_1, i_2, a_2, \cdots, a_{q-1}, i_q$ donde $i_1 = i_q$, es decir, el extremo inicial de la cadena es el extremo final de ésta, también puede escribirse como $a_1, a_2, \cdots, a_{q-1}$.

Un *circuito Hamiltoniano (ciclo Hamiltoniano)* es uno que utiliza todos los nodos de la gráfica. Se dice que si un circuito (ciclo) pasa por cada arista exactamente una sola vez, se conoce como circuito Euleriano (ciclo Euleriano).

Un ejemplo de circuito hamiltoniano se encuentra en la figura 1.3 dado por el conjunto de aristas $(x_1, x_2), (x_2, x_4), (x_4, x_5), (x_5, x_3), (x_3, x_1)$.

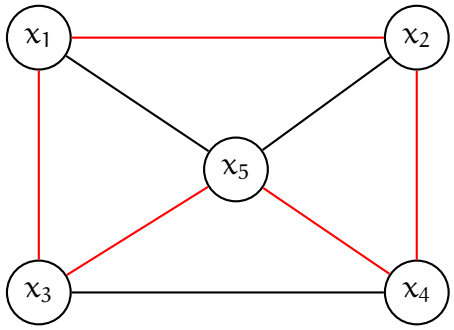


Figura 1.3: Un circuito hamiltoniano es: $x_1 \rightarrow x_2 \rightarrow x_4 \rightarrow x_5 \rightarrow x_3 \rightarrow x_1$

Una gráfica $G = [X, A]$ es *conexa* si para cualesquiera dos vértices, existe una cadena que los une. De lo contrario se habla de una gráfica disconexa.

Una gráfica es *completa* si $\forall\ i, j \in X$ existe un arco o arista $(i, j) \in A$.

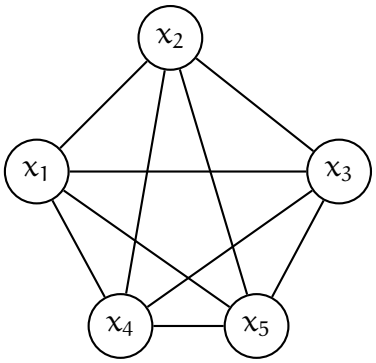
Una gráfica $G = [X, A]$ es *bipartita* si su conjunto de nodos se puede ver como una partición de dos conjuntos tal que cumplen:

i) $X = X^a \cup X^b$

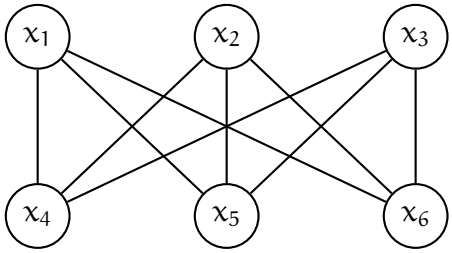
ii) $X^a \cap X^b = \varnothing$

iii) $\forall\ (i,j) \in A$ se cumple que $i \in X^a$ y $j \in X^b$ o viceversa.

A continuación se muestran dos ejemplos: la figura 1.4a es una gráfica completa de 5 vértices porque existe una arista para cualesquiera par de vértices que se tomen; la figura 1.4b es una gráfica bipartita, ya que la partición dada por $V = V_1 \cup V_2$, donde V_1 representa a los 3 vértices de arriba y V_2 los 3 vértices de abajo, logra hacer una bipartición del conjunto V de la gráfica.



(a) Gráfica completa de 5 nodos y 10 aristas.



(b) Gráfica bipartita con $|X^a| = 3$ y $|X^b| = 3$.

Figura 1.4: Ejemplos de gráfica completa y gráfica bipartita.

¿Por qué es interesante saber el número de ciclos hamiltonianos?

Para el problema del agente viajero las soluciones son representadas como ciclos hamiltonianos por lo que se analiza el cálculo del número de ciclos hamiltonianos para una gráfica completa $G = [X, A]$ con $|X| = n$. Inicialmente se fija el primer vértice i_1 que sirve como punto de inicio para el ciclo hamiltoniano, después se observa que de este se pueden visitar los $n-1$ vértices restantes por la propiedad de la gráfica, así que se selecciona el siguiente vértice i_2 ahora se pueden visitar los $n-2$ vértices restantes, entonces para cuando se seleccione el vértice i_{n-1} solamente puede visitarse el último vértice i_n y al tomar el vértice final i_n ya se habrán visitado todos los vértices. Por lo tanto, el número de ciclos hamiltonianos posibles en G son: $(n-1)(n-2)(n-3)...3\cdot2\cdot1$, es decir, $(n-1)!$, y dado que se puede recorrer en sentido inverso se tienen la mitad de posibilidades, por lo tanto el número de ciclos hamiltonianos que hay en una gráfica completa no dirigida son: $\frac{(n-1)!}{2}$.

¿Cómo trabajar con un problema de optimización?

Una vez que se tiene un problema de optimización con el que se trabajará, lo que sigue es pensar cómo definir varias componentes como las variables que lo definen, el objetivo que se debe alcanzar y las restricciones, después comenzar a pensar en la estrategia que se utilizará para encontrar soluciones. Los siguientes puntos mencionan lo anterior:

- **Formulación del problema:** en este paso se declaran las partes del problema identificando: el objetivo del problema, las restricciones y las variables de decisión. Esta primera parte puede ser imprecisa, debido a que aún faltan otros puntos de revisión.
- **Modelación del problema:** este es un paso importante y posiblemente abstracto, ya que se formula el problema como un modelo matemático, puede inspirarse de algún modelo similar ya existente. Los modelos matemáticos a resolverse son simplificaciones de problemas de la realidad y uno de los más utilizados y conocidos en la programación matemática es el de **programación lineal**, el cual se formula en la ecuación 1.4. Esta formulación matemática fue propuesta por el físico y matemático George Dantzig en 1947.

$$\begin{array}{ll} \text{Min (Max)} & c \cdot x \\ \text{sujeto a:} & A \cdot x \geq b \quad \text{o bien} \quad \leq \text{o} = \\ & x \geq 0 \end{array} \tag{1.4}$$

- **Implementación de búsqueda de la solución:** las soluciones generadas deben ser aprobadas de acuerdo a las restricciones del modelo planteado y formulado. Se define un algoritmo de optimización tal que al repetir el proceso se logre alcanzar la mejor de las soluciones factibles evaluadas.

Entonces, un problema de optimización es aquel cuya solución implica explorar el conjunto de soluciones candidatas y seleccionar la que mejor cumple los objetivos del problema. Así, de manera general se plantea un problema de optimización como:

$$\begin{array}{ll} \text{Optimizar} & c(s) \\ \text{sujeto a:} & \\ & s \in F \end{array}$$

Donde F es el espacio de todas las posibles soluciones asociadas al problema y c es una función definida en $\mathbb{R}$ conocida como la *función objetivo*, tal que a cada solución $s \in F$ se le asocia un

costo o valor $c(s)$. Si el objetivo del problema es minimizar entonces, se dice que una solución s^* es óptimo global si $c(s^*) \leq c(s)\ \forall s, s^* \in F$. De manera similar si el objetivo es maximizar, s^* es solución óptimo global si $c(s^*) \geq c(s)\ \forall s, s^* \in F$.

Los problemas de optimización utilizan variables continuas o discretas; los que emplean variables discretas son llamados *problemas de optimización combinatoria*.

1.3. Formulación matemática del Problema del Agente Viajero.

Existen distintas formas de plantear el PAV matemáticamente. A continuación se describen algunas de estas [11].

Sea G = [X,A] una gráfica completa (dirigida o no dirigida) y sea $\mathbb{F}$ la familia de ciclos hamiltonianos (tours) en G. Cada arista $a_i \in A$ tiene un costo asociado representado en la matriz $C = (c_{ij})_{nxn}$, llamada matriz de costos o matriz de distancias, donde cada entrada c_{ij} corresponde al costo asociado de la arista que une al vértice i con el vértice j. Entonces, el objetivo del PAV es encontrar el ciclo hamiltoniano (tour) en G tal que su distancia total sea la menor del conjunto $\mathbb{F}$.

El PAV puede verse como un problema de permutaciones, denotando al conjunto $\mathbb{P}_n$ a la colección de permutaciones del conjunto de ciudades $1, 2, \cdots, n$. Entonces, el objetivo del PAV es encontrar una permutación $\pi = (\pi(1), \pi(2), \cdots, \pi(n))$ en $\mathbb{P}_n$ tal que se minimice la distancia asociada esto se expresa en la ecuación 1.5:

$$c_{\pi(n)\pi(1)} + \sum_{i=1}^{n-1} c_{\pi(i)\pi(i+1)} \tag{1.5}$$

Un ejemplo de la ecuación 1.5 se muestra en la figura 1.5 con el conjunto de ciudades {1,2,3,4,5} con sus respectivas distancias d_{ij}.

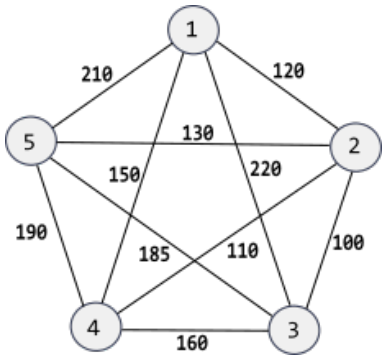


Figura 1.5: Instancia del problema del agente viajero con 5 ciudades. Generado en *Inkscape*.

Entonces, una solución puede verse como la permutación dada por $(\pi(1), \pi(3), \pi(4), \pi(2), \pi(5))$ con las respectivas distancias $c_{13} = 220, c_{34} = 160, c_{42} = 185, c_{25} = 130$, con una distancia total de:

$$c_{13} + c_{34} + c_{42} + c_{25} + c_{51} = 905 \quad \text{unidades}$$

Otra forma de plantear el problema es como uno de *permutaciones cíclicas*. Definimos al conjunto $\mathbf{C}_n$ como la colección de permutaciones cíclicas de las n ciudades. Entonces, de entre todas las posibles permutaciones cíclicas factibles del problema, se busca encontrar la que tenga una menor distancia. Es decir, se minimiza la ecuación 1.6:

$$\sum_{i=1}^{n} c_{i\sigma(i)} \tag{1.6}$$

Donde $\sigma(i)$ es el sucesor de la ciudad i en el tour resultante, para $i = \{1, 2, \cdots, n\}$. Por ejemplo, para $n = 5$ se genera el siguiente tour $(\sigma(1), \sigma(2), \sigma(3), \sigma(4), \sigma(5)) = (2, 4, 5, 3, 1)$, así la permutación cíclica resultante es $(1, 2, 4, 3, 5, 1)$, cuya distancia es

$$c_{1,\sigma(1)} + c_{2,\sigma(2)} + c_{3,\sigma(3)} + c_{4,\sigma(4)} + c_{5,\sigma(5)} = 785 \quad \text{unidades,}$$

Al final, para todas las formulaciones se observa que el objetivo es minimizar la distancia de entre todas las posibles soluciones asociadas al problema.

Dependiendo de la estructura de las instancias del PAV, la matriz de distancias puede comportarse de distintas maneras: sea C la matriz de distancias, si C es simétrica el PAV se define como un problema simétrico, si C no es simétrica se define como un problema asimétrico. De aquí

obtenemos las siguientes afirmaciones:

i) El problema del agente viajero es simétrico (STSP), si $d_{ij} = d_{ji}\ \forall (i,j) \in A$

ii) El problema del agente viajero es asimétrico (ATSP), si $\exists\ (i,j) \in A \mid d_{ij} \neq d_{ji}$

iii) El problema del agente viajero es triangular ($\triangle$ TSP), si $d_{ik} \leq d_{ji}\ \forall i,j,k$

También pueden aplicarse otras variantes al problema, entre los cuales:

i) El problema del agente múltiple **m-TSP** es una generalización del PAV, que consiste en determinar un conjunto de rutas para *m* vendedores que inician y finalizan de una ciudad origen [5].

ii) El problema del agente viajero multicolor **MTSP** utiliza una gráfica completa cuyo conjunto de nodos se dividen en k-**subconjuntos** que son identificados con colores [15].

iii) El problema del doble agente viajero con múltiples pilas, **DTSPMS** (The double traveling salesman problem with multiple stacks). El cual es un problema de encaminamiento de vehículos que consiste en encontrar los recorridos de longitud total mínima en dos redes separadas, una para recogidas y otra para entregas; dado un conjunto de pedidos, éstos deben ser enviados desde un punto de recogida y hasta un punto de entrega [10].

1.3.1. Formulación matemática del PAV

Formulación de programación entera del PAV

Para una instancia de n ciudades se tiene:

- $X = \{1, 2, \cdots, n\}$ conjunto de vértices que representan las distintas ciudades.
- $A = \{(i,j) : i,j \in X\}$ es el conjunto de aristas definidas entre cada par de ciudades i,j.
- Se define la función $c : A \rightarrow \mathbb{R}$ para todo elemento de A tal que $c_{ij} =$\{distancia asociada para (i,j) con $i,j \in X$\}.

Y las variables de decisión se definen como:

$$x_{ij} = \begin{cases} 1 & \text{si se viaja de la ciudad } i \text{ a la ciudad } j \\ 0 & \text{si no se viaja de la ciudad } i \text{ a la ciudad } j \end{cases}$$

El siguiente modelo lineal entero binario fue propuesto por Dantzig, Fulkerson y Johnson [11], esta formulación es de gran importancia, ya que inicialmente solo se analizaba que, para tener una solución factible bastaba asegurar que exista una arista que una a la ciudad i con una ciudad j y de la misma manera que exista una arista que una a la ciudad j con una ciudad i. Sin embargo, esto no era suficiente, pues se observó que los subciclos podían ser candidatos como soluciones factibles, así que se vio la necesidad de añadir una restricción más la cual eliminará estos casos.

$$\text{Minimizar} \qquad \sum_{i=1}^{n} \sum_{j=1}^{n} c_{ij} x_{ij}$$

sujeto a:

$$\sum_{i \neq j, i=1}^{n} x_{ij} = 1; \qquad j = 1, \ldots, n \tag{1.7a}$$

$$\sum_{i \neq j, j=1}^{n} x_{ij} = 1; \qquad i = 1, \ldots, n \tag{1.7b}$$

$$\sum_{i,j \in Q} x_{ij} \leq |Q| - 1; \quad \forall\ Q \subseteq X \text{ y } 2 \leq |Q| \leq n-1 \tag{1.7c}$$

$$x_{ij} = 0, 1 \qquad \forall\ (i,j) \in A$$

La restricción 1.7a asegura que para cada ciudad i hay una única arista adyacente que la une con la ciudad j, la restricción 1.7b refleja que para cada ciudad j hay una única arista que la une con una ciudad i.

La restricción 1.7c indica que para cada subconjunto Q de vértices que no consideren al vértice 1 no son soluciones, es decir, no se pueden formar subciclos con un número de vértices menor o igual a $n-1$ (sin pérdida de generalidad no se considera el vértice 1).

Para ilustrar la restricción 1.7c se tiene la figura 1.6.

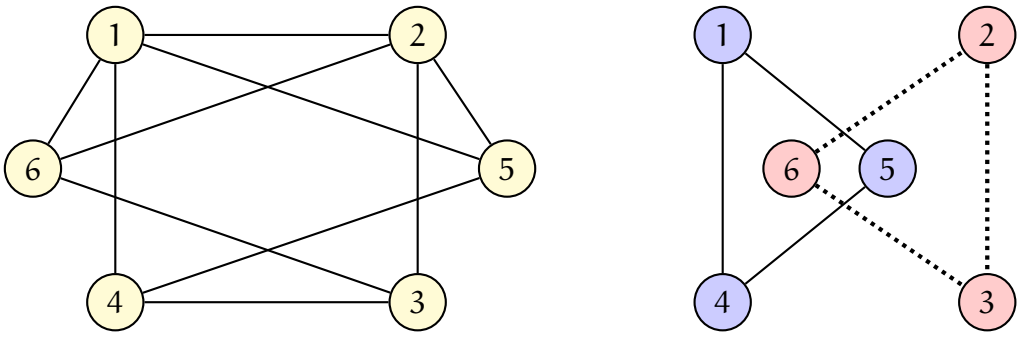


Figura 1.6: Del lado izquierdo se tiene la gráfica original mientras que del lado derecho se encuentra una solución con dos subciclos, definidos por los subconjuntos de vértices: $\{1,4,5\}$ y $\{2,3,6\}$. Ambos cumplen la condición de visitar cada ciudad una sola vez. Sin embargo, la restricción 1.7c no se cumple para el subconjunto $\{2,3,6\}$ porque $x_{23}+x_{36}+x_{62}=3$ y esto no es menor o igual a 2.

Se observa que para el subconjunto de ciudades $Q=\{2,3,6\}$ se cumple que:

$$x_{23}=x_{36}=x_{62}=1$$

Veamos como se traducen las restricciones 1.7a y 1.7b para el subciclo. Por ejemplo, para la ciudad 2 las aristas incidentes al vértice son $(2,3)$ y $(6,2)$, entonces $x_{23}=x_{62}=1$ lo que significa que las demás variables correspondientes a los vecinos de 2 valen cero, ya que no se recorren en la solución. Como las variables son binarias entonces las restricciones correspondientes son:

- $i=2$ se tiene que $x_{23}=1$ entonces $x_{21}=x_{25}=x_{26}=0$ para $\{(2,1),(2,5),(2,6)\}\in A$.
- $j=6$ se tiene que $x_{62}=1$ entonces $x_{61}=x_{63}=0$ para $\{(6,1),(6,2),(6,3)\}\in A$.

Análogamente se analiza de ésta manera para las ciudades 3 y 6. Entonces las restricciones para el subciclo $\{2,3,6\}$ son:

La restricción 1.7a:

$$\begin{aligned} x_{21}+x_{23}+x_{25}+x_{26} &= 1 \quad \text{para} \quad i=2\\ x_{32}+x_{34}+x_{36} &= 1 \quad \text{para} \quad i=3\\ x_{61}+x_{62}+x_{63} &= 1 \quad \text{para} \quad i=6 \end{aligned}$$

La restricción 1.7b:

$$\begin{aligned} x_{12} + x_{32} + x_{52} + x_{62} &= 1 \quad \text{para} \quad j = 2 \\ x_{23} + x_{43} + x_{63} &= 1 \quad \text{para} \quad j = 3 \\ x_{16} + x_{26} + x_{36} &= 1 \quad \text{para} \quad j = 6 \end{aligned}$$

De igual forma se pueden verificar para el subconjunto $\{1, 4, 5\}$.

Sin embargo, al verificar la restricción 1.7c con el subconjunto que no contiene al nodo 1, $Q = \{2, 3, 6\}$ se tiene que:

$$x_{23} + x_{36} + x_{62} \leq 2 \quad \text{con} \quad x_{23} = x_{36} = x_{62} = 1$$

Esto no es cierto porque la cardinalidad de Q es 3 y se tendría que $3 \leq 3 - 1$ lo cual es una contradicción.

Entonces esta formulación permite la eliminación de subciclos, pero hay un inconveniente con el número de restricciones resultantes de la condición 1.7c, dado que Q representa todos los posibles subconjuntos de vértices generados con al menos dos elementos que no contengan al vértice 1. Esto significa que habrá un número exponencial de subconjuntos e implica una cantidad exponencial de restricciones las cuales deben verificarse. Entonces, puede verse que al aumentar el número de ciudades se tienen tantas restricciones como el número de subconjuntos Q del conjunto total de nodos, es decir, por lo menos en el peor de los casos se tendrán 2^n restricciones.

En 1960 Miller, Tucker y Zemlin (MTZ) [17] proponen otra manera de eliminar los subciclos y muestran que a lo más hay $(n-1)^2$ restricciones adicionales y $n-1$ variables extras y no se utilizan subconjuntos Q. Reemplazando la restricción 1.7c por la restricción 1.8,

$$u_i - u_j + n x_{ij} \leq n - 1, \qquad 2 \leq i \neq j \leq n \tag{1.8}$$

Las variables auxiliares $u_i \in \mathbb{R}$ representan la posición que ocupa la ciudad i en la solución, es decir, si $u_i = k$ entonces la ciudad i ocupa la posición k en la solución. La restricción 1.8 también puede escribirse como: $n x_{ij} - (u_j - u_i) \leq n - 1$.

En la figura 1.7 se explica la interpretación visual de las variables u_i.

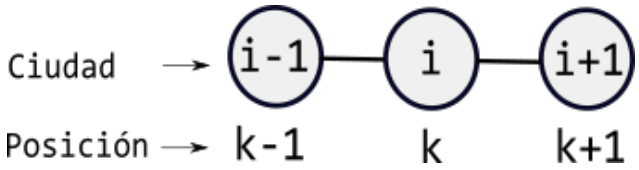


Figura 1.7: Si la ciudad i tiene la posición k en la solución, entonces $u_i = k$. La ciudad anterior a i ocupa la posición $k-1$ y la ciudad siguiente de i ocupa la posición $k+1$, por lo tanto $u_{i-1} = k-1$ y $u_{i+1} = k+1$ respectivamente. Generado en *Inkscape*.

Además, sin pérdida de generalidad puede decirse que las variables u_i son enteras, $u_i \in \mathbb{Z}^+$, y se cumple que $u_i \neq u_j \; \forall \, i \neq j$, lo que quiere decir que dos ciudades no pueden estar en la misma posición, entonces $u_i - u_j \neq 0 \; \forall \, i \neq j$. Por lo tanto se puede verificar lo siguiente,

$$u_j - u_i = \begin{cases} u_j - u_i \geq 1 & \text{si } u_j > u_i \\ u_j - u_i \leq -1 & \text{si } u_j < u_i \end{cases}$$

Para no comparar ciclos iguales se establece una ciudad de inicio, ya que en un ciclo hamiltoniano elegir la ciudad de inicio y fin puede ser cualquiera y no altera la distancia. Sea $u_1 = 1$, entonces $2 \leq u_i \leq n$ para $i = \{2, 3 \cdots, n\}$.

Usando el ejemplo 1.6 y recordando que el subciclo dado por $\{2, 3, 6\}$ cumple las restricciones 1.7a y 1.7b, ahora se analiza que pasa con la restricción 1.8,

$$\begin{aligned} (u_3 - u_2) + nx_{23} &\leq n-1 \\ (u_6 - u_3) + nx_{36} &\leq n-1 \\ (u_2 - u_6) + nx_{62} &\leq n-1 \end{aligned}$$

Sumando estas restricciones se tiene que $n \leq n-1$ lo cual no es cierto. Entonces, se comprueba que este subciclo no es válido para el problema.

A continuación se demuestra que en efecto estas restricciones definen al PAV.

Proposición 1. *El conjunto de ecuaciones 1.7a, 1.7b y 1.8 define al PAV.*

Demostración. A continuación se demostrará que, si una solución es un ciclo hamiltoniano entonces es una solución factible para el PAV.

Sea S una solución factible del problema, la cual debe ser un ciclo hamiltoniano, entonces se cumple que las variables de decisión toman valores $x_{ij} = \{1, 0\}$ y se satisface que:

$$\sum_{i \neq j} x_{ij} = 1 \quad \text{para} \quad j = 1, 2, \cdots, n \quad \text{y} \quad \sum_{j \neq i} x_{ij} = 1 \quad \text{para} \quad i = 1, 2, \cdots, n.$$

Lo cual indica que existe una arista que une la ciudad i con la ciudad j y del mismo modo hay una arista que une la ciudad j con la ciudad i. Por lo tanto, puede decirse que para cada arista del ciclo hamiltoniano S, se le asigna un único vértice que es extremo final y otro que es extremo inicial. Supóngase que en vez de formarse un único ciclo hamiltoniano se forman dos o más subciclos. Entonces, se elige un subciclo que no contenga al vértice 1, $S' = \{(i_2, i_3), \cdots, (i_{k-1}, i_k), (i_k, i_2)\}$ donde el número de aristas de S' es igual a k con $2 \leq k \leq n-1$ para todo $i_j \neq 1$. Si esta solución es factible se cumple que $u_i - u_j + n x_{ij} \leq n - 1 \ \forall \ 2 \leq i \neq j \leq n$, y además las x_{ij} correspondientes a esta solución son iguales a 1, por lo tanto se tiene lo siguiente,

$$\begin{aligned}
(u_{i_2} - u_{i_3}) + n x_{i_2 i_3} &\leq n - 1 \\
(u_{i_3} - u_{i_4}) + n x_{i_3 i_4} &\leq n - 1 \\
&\vdots \\
(u_{i_k} - u_{i_2}) + n x_{i_k i_2} &\leq n - 1
\end{aligned}$$

Se suman las restricciones anteriores obteniendo la ecuación 1.9

$$\sum_{(i,j) \in S'} u_i - u_j + n x_{ij} \leq k(n-1) \tag{1.9}$$

Notemos que para la parte de la suma $u_i - u_j$ se cumple que

$$\sum_{(i,j) \in S'} u_i - u_j = 0 \tag{1.10}$$

Ya que cada u_i aparece una vez sumándose y una vez restándose y, además, para las aristas de S' se satisface que $x_{ij} = 1$, entonces para la otra parte se tiene

$$\sum_{(i,j)\in S'} nx_{ij} = kn \tag{1.11}$$

Y por lo tanto, de 1.10 y 1.11 se obtiene la ecuación 1.12,

$$kn = 0 + kn = \sum_{(i,j)\in S'} u_i - u_j + \sum_{(i,j)\in S'} nx_{ij} = \sum_{(i,j)\in S'} u_i - u_j + nx_{ij} \leq k(n-1) \tag{1.12}$$

Es decir, $kn \leq k(n-1)$, lo cual es absurdo, ya que $k > 0$ y no se cumple que $n \leq n-1$. Por lo tanto, una solución factible debe ser necesariamente un ciclo hamiltoniano (es decir, es un ciclo que contiene todos los vértices de la gráfica). Y se considera una solución no factible cuando contenga subciclos.

Ahora veamos que la solución factible S satisface las restricciones 1.7a y 1.7b, para esto aseguraremos la existencia de variables u_i cuyos valores satisfacen la restricción 1.8 por lo que S es solución factible del problema. Fijando al vértice 1 como $u_1 = 1$, es decir, la ciudad 1 es la primera en visitarse, sea $u_i = k$ si la i-ésima ciudad se visita el k-ésimo lugar, para $i \neq 1$. Entonces, si $x_{ij} = 1$, se cumple

$$u_i - u_j + nx_{ij} = k - (k+1) + n = n - 1$$

Y si $x_{ij} = 0$, se satisface

$$u_i - u_j + nx_{ij} \leq n - 2 < n - 1$$

Ya que $2 \leq u_i \leq n$, $\forall i \in \{2, \cdots, n\}$.

Y por lo tanto, un ciclo hamiltoniano es una solución factible para el PAV. □

De esta manera se concluye que el problema está bien definido por las ecuaciones 1.7a, 1.7b y 1.8.

Formulación de programación entera para el PAV simétrico

El problema del agente viajero simétrico consiste en que el costo o distancia de la ciudad i a la ciudad j es el mismo que de la ciudad j a la ciudad i, es decir, se cumple que $c_{ij} = c_{ji}$ para $1 \leq i \neq j \leq n$. La primera restricción se conoce como *restricción de grado* y la segunda funciona para eliminar la posibilidad de subciclos como se mencionó anteriormente.

$$
\begin{array}{lll}
\text{Minimizar} & \sum_{i=1}^{n} \sum_{j=1}^{n} c_{ij} x_{ij} & \\
\text{sujeto a:} & & \\
 & \sum_{i \neq j} x_{ij} + \sum_{i \neq j} x_{ij} = 2 & \forall\, i, j \\
 & \sum_{i,j \in Q} x_{ij} \leq |Q| - 1; & \forall\ Q \subseteq X \text{ y } 2 \leq |Q| \leq n-1 \\
 & x_{ij} = 0, 1 & \forall\, i,\ j \in A
\end{array}
\tag{1.13}
$$

Capítulo 2

Optimización y complejidad computacional

El concepto de *optimizar* hace referencia a buscar "lo mejor" dependiendo el contexto, por ejemplo, si se busca optimizar el tiempo para llegar de un punto de salida a un punto de llegada se establecerán decisiones como recorrer un menor número de calles, pasar por el menor número de semáforos, etcétera.

En particular para el PAV el objetivo es minimizar la distancia de entre todas las posibles soluciones dadas en el espacio de búsqueda, que son los posibles ciclos hamiltonianos en una

gráfica. Existen algoritmos que permiten resolver el PAV, sin embargo el número de ciudades o el tamaño de la instancia hace difícil encontrar una buena solución en un corto tiempo.

Así como este problema existen otros que son difíciles de resolver en el sentido de encontrar la mejor solución de una instancia con una gran cantidad de datos, ya sea en el número de variables o restricciones. Para eso se han implementado algoritmos que encuentran la mejor solución (local o global) en un tiempo computacional razonable.

2.1. Instancia de un problema

Definición 2.1.1. *Una instancia de un problema se define como la dupla (F,c), donde F es el conjunto de soluciones factibles y c es la función de costo definida como*

$$c : F \to \mathbb{R}$$

Donde a cada elemento del conjunto de soluciones factibles se le asocia un valor.

Entonces debe encontrarse una $x \in F$ *para la cual se cumpla*

$$c(x) \leq c(y) \quad \forall\ y \in F$$

El punto x es llamado solución óptima global de la instancia de un problema de minimización.

En otras palabras, un problema de optimización está comprendido por una colección de instancias que son generadas de manera similar, donde cada instancia representa datos de entrada de un problema.

Instancias para el PAV

Una instancia del PAV está constituida por un conjunto de n ciudades donde el objetivo es encontrar de entre todos los elementos de F el de menor distancia, con

$$F = \{\text{todas las permutaciones } \pi \text{ cíclicas de n ciudades}\}$$

Donde una permutación cíclica representa un ciclo o tour, donde $\pi(i)$ es la ciudad visitada después de la ciudad j, $i = 1, \cdots, n$. Por lo tanto, el costo representado por alguna solución se calcula con la ecuación 1.5.

2.1.1. Caracterización de problemas e instancias

Para cada elemento s del conjunto F de soluciones factibles se calcula el costo $c(s)$ y se asocian implícitamente los algoritmos A_F y A_c, respectivamente. A_F es el algoritmo de prueba encargado de determinar si, para un elemento l del espacio de soluciones, se cumple que $l \in F$ o $l \notin F$, y se construye el conjunto de soluciones factibles caracterizado por los parámetros en P_F. Por otro lado, dada una solución factible $s \in F$ y un conjunto de parámetros P_c, A_c es el algoritmo asociado a la función de costo $c(s)$. Entonces cada problema de optimización combinatoria está definido por un algoritmo de reconocimiento A_F y un algoritmo de función de costo A_c, mientras que cada instancia de un problema tiene asociado los conjuntos de parámetros P_F y P_c.

2.1.2. Un problema tiene tres versiones

Dado un problema de optimización combinatoria se dice que tiene tres versiones.

Definición 2.1.2. *(Versión de optimización) Dadas las representaciones para los conjuntos de parámetros P_F y P_c para los algoritmos A_F y A_c respectivamente de un problema de optimización combinatoria. Encontrar una solución óptima factible.*

Si únicamente se pretende encontrar el costo de una solución óptima factible, entonces se relaja la definición anterior obteniendo la versión de evaluación del problema, definida a continuación.

Definición 2.1.3. *(Versión de evaluación) Dadas las representaciones para los conjuntos de parámetros P_F y P_c para los algoritmos A_F y A_c respectivamente de un problema de optimización combinatoria. Encontrar el costo de una solución óptima factible.*

Bajo el supuesto de que se trabaja con un algoritmo A_c de tiempo polinomial, la versión de evaluación no puede ser más difícil que la versión de optimización, esto debido a que al resolver la versión de optimización ya se conoce la solución óptima factible y por lo tanto su costo asociado puede ser calculado en tiempo polinomial mediante la función de costo asociada al algoritmo A_c.

La siguiente versión está dada para un problema de minimización la cual tiene asociada una pregunta en donde la respuesta es: "sí" o "no".

Definición 2.1.4. *(Versión de decisión) Dadas las representaciones para los conjuntos de parámetros* P_F *y* P_f *para los algoritmos* A_F *y* A_c *respectivamente para un problema de optimización combinatoria y* B *un número entero que representa una cota del problema, ¿existe una solución factible* $s \in F$ *tal que* $c(s) \leq B$*?*

Análogamente para un problema de maximización se busca una solución factible $s \in F$ tal que $c(s) \geq B$. La versión de decisión no debe ser más difícil que la versión de evaluación porque una vez calculado el costo de una solución óptima factible en la versión de evaluación, puede ser comparada con cualquier valor B que de respuesta a la versión de decisión. De esta manera se establece una jerarquía de versiones para un problema.

¿Cuáles son las tres versiones para el PAV?

Para el problema del agente viajero el conjunto de parámetros P_F establece factibilidad de las soluciones. Sea $G = [X, A]$ una gráfica completa, una solución es factible si es un ciclo hamiltoniano de G de esta manera definimos $F = \{$todos los posibles ciclos hamiltonianos de $G\}$. El conjunto de parámetros P_c está definido por las distancias d_{ij} calculadas o establecidas entre cualquier par de ciudades $i, j \in X$. El algoritmo de reconocimiento A_F verifica que dada una solución s pertenece al conjunto F. Y el algoritmo de función de costo A_c suma las distancias asociadas de la solución $s \in F$ asignándole un valor real $c(s) =$ distancia del ciclo hamiltoniano. Entonces las tres versiones del PAV quedan de la siguiente manera:

i) Versión de optimización: dada una gráfica completa $G = [X, A]$ con distancias no negativas d_{ij} entre cualquier par de vértices. Encontrar el ciclo hamiltoniano de menor longitud.

ii) Versión de evaluación: dada una gráfica completa $G = (X, A)$ con distancias no negativas d_{ij} entre cualquier par de vértices. Calcular la distancia del ciclo hamiltoniano de menor longitud.

iii) Versión de decisión: dada una gráfica completa $G = (X, A)$ con distancias no negativas d_{ij} entre cualquier par de vértices y un entero B ¿Existe algún ciclo hamiltoniano con distancia menor o igual a B?

2.1.3. Espacio de búsqueda y vecindades

Una vecindad de una solución $s \in F$ se define como el conjunto de soluciones *cercanas* a s que está conformado por cualquier subconjunto $N(s) = \{s_1, ..., s_p\}$ en F. La manera de explorar a estas soluciones cercanas $s_i \in N(s)$ es mediante *movimientos*.

Se dice que dos soluciones $s, s' \in N(s)$ son vecinas si difieren en pocos elementos o movimientos. Un movimiento consiste simplemente en cambiar uno o varios elementos de s. En otras palabras, una solución es vecina de otra después de aplicar ciertos movimientos a la solución original y por lo general $s \in N(s')$ siempre que $s' \in N(s)$. Entonces el espacio de búsqueda queda definido por el conjunto de todas las posibles soluciones de un problema específico.

En un problema de optimización combinatoria dada una solución inicial se implementan estratégicamente funciones de movimiento con el fin de explorar la mayor cantidad del espacio de búsqueda del problema y encontrar soluciones de mejor calidad. Esto se puede convertirse en un desafío al que se enfrenta en la implementación de la búsqueda local asociada al problema [22].

2.1.4. Algoritmos de optimización

Existen dos ramas de algoritmos de optimización: determinísticos y estocásticos. En esta sección se explica a grandes rasgos estas dos categorías y se da un panorama general de la ramificación.

Algoritmos determinísticos

Los algoritmos deterministas producen sobre una entrada determinada los mismos resultados siguiendo los mismos pasos del cálculo. Entonces el comportamiento del algoritmo está determinado por su entrada y no dependen de ningún elemento de aleatoriedad o incertidumbre.

Algoritmos aleatorios

Los algoritmos aleatorios actúan de manera contraria a los algoritmos determinísticos, producen un resultado diferente cada vez que se pasa una entrada. Estos tipos de algoritmos exploran

de manera aleatoria algunos elementos del conjunto de soluciones, por lo que en cada iteración se puede obtener un resultado distinto que puede o no tener un mejor valor que el anterior.

Si una heurística o metaheurística es determinista, significa que resuelve un problema tomando decisiones determinísticas, por ejemplo: búsqueda local, búsqueda tabú. Y decimos que una heurística o metaheurística es aleatoria si se aplican reglas aleatorias durante la búsqueda, por ejemplo: recocido simulado, algoritmos evolutivos.

A continuación se muestran en las figuras 2.1 y 2.2 la división de algunos algoritmos conocidos.

Figura 2.1: Diagrama de algoritmos determinísticos. Generado en *Lucidchart*. Fuente: [16]

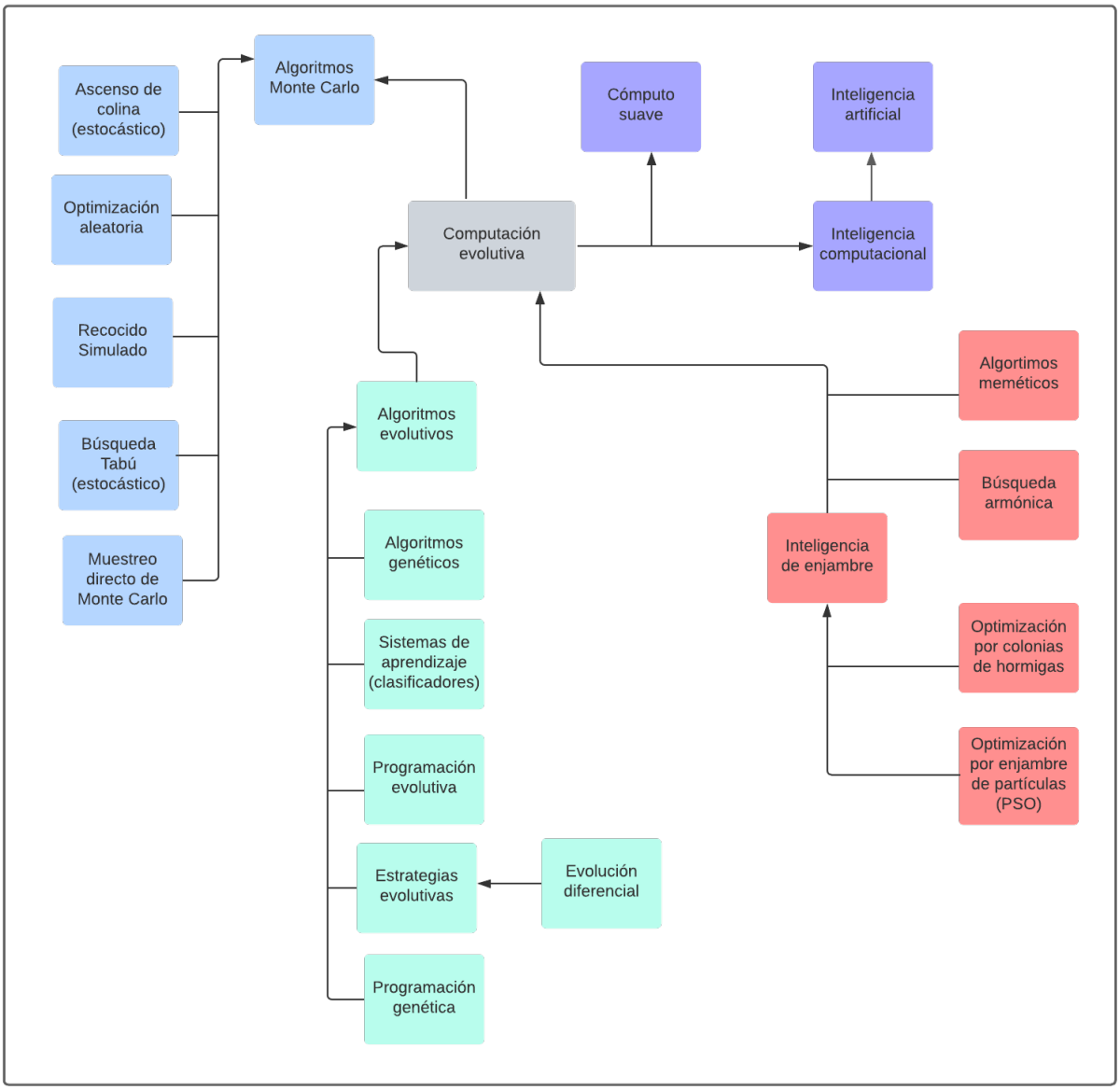


Figura 2.2: Diagrama de algoritmos aleatorios. Generado en *Lucidchart*. Fuente: [16]

2.2. Teoría de la complejidad

La idea intuitiva de la complejidad de un algoritmo es analizar que tan "eficiente o bueno es el algoritmo" analizando el número de iteraciones o número de pasos que requiere para ejecutarse.

Entonces para saber que tan bueno o malo es un algoritmo dada cualquier instancia, puede contarse el número de iteraciones que realiza el algoritmo. Pero sin importar que tan bueno o malo sea, se analiza su funcionamiento con todas las posibles instancias que se le envíen. Para esto se definen los siguientes tres posibles casos a analizar;

Definición 2.2.1. *La complejidad del algoritmo en el peor de los casos es la función definida por el número máximo de pasos para cualquier instancia de tamaño* $\mathfrak{n}$.

Definición 2.2.2. *La complejidad del algoritmo en el mejor de los casos es la función definida por el mínimo número de pasos dados para cualquier instancia de tamaño* $\mathfrak{n}$.

Definición 2.2.3. *La complejidad promedio del algoritmo es la función definida por el número promedio de pasos en todos los casos de tamaño* $\mathfrak{n}$.

Así que para un algoritmo se analiza cada uno de los casos anteriores de complejidades de tiempo sobre todos los tamaños de las posibles instancias, cada caso puede utilizar una función numérica distinta y esto puede convertirse en un trabajo difícil, para esto se utiliza la notación de la $\mathcal{O} - \mathfrak{grande}$ o bien $\mathcal{O} - \mathfrak{grandota}$ o mejor conocida en inglés como $\mathfrak{Big} - \mathcal{O}$.

La notación $\mathcal{O} - \mathfrak{grande}$ simplifica muchos cálculos y análisis sin generar un impacto en la comparación de algoritmos. Por ejemplo, ignora las constantes multiplicadas, si $\mathfrak{f}(\mathfrak{n}) = 2\mathfrak{n}$ y $\mathfrak{g}(\mathfrak{n}) = \mathfrak{n}$ ambas son iguales para el análisis de $\mathcal{O} - \mathfrak{grande}$. A continuación se mencionan algunas definiciones relacionadas a la complejidad. [24]

Definición 2.2.4. *($\mathcal{O} - \mathfrak{grande}$) Un algoritmo tiene complejidad* $\mathfrak{f}(\mathfrak{n}) = \mathcal{O}(\mathfrak{g}(\mathfrak{n}))$, *si existen las constantes positivas* $\mathfrak{n}_0$ *y* $\mathfrak{c}$ *tales que* $\forall\ \mathfrak{n} \geq \mathfrak{n}_0$ *cumplen que* $\mathfrak{f}(\mathfrak{n}) \leq \mathfrak{c} \cdot \mathfrak{g}(\mathfrak{n})$.

En otras palabras, si se encuentra una constante $\mathfrak{c}$ adecuada, entonces se logra hacer $\mathfrak{cg}(\mathfrak{n})$ una cota superior para $\mathfrak{f}(\mathfrak{n})$, para valores de $\mathfrak{n}$ lo suficientemente grandes. La notación $\mathcal{O} - \mathfrak{grande}$ puede ser utilizada para denotar la complejidad del tiempo o espacio de un algoritmo.

Definición 2.2.5. *(Algoritmo de tiempo-polinomial.) Un algoritmo es de tiempo polinomial si su complejidad es* $\mathcal{O}(\mathfrak{p}(\mathfrak{n}))$, *donde* $\mathfrak{p}(\mathfrak{n})$ *es una función polinomial de n.*

Una función polinómica de grado *k*, se define como:

$$p(n) = a_k \cdot n^k + ... + a_j \cdot n^j + ... + a_1 \cdot n + a_0$$

donde $a_k > 0$ y $a_j \geq 0 \ \forall 1 \leq j \leq k-1$. Si se tuviera un algoritmo de tiempo polinomial de grado k, se denotaría como $\mathcal{O}(n^k)$.

Definición 2.2.6. *(Algoritmo de tiempo-exponencial.) Un algoritmo es de tiempo exponencial si su complejidad es $\mathcal{O}(c^n)$, donde c es un número en los reales, estrictamente mayor a 1.*

La notación $\mathcal{O} - grande$ es parte de una familia de notaciones como $\Omega - grande$ y $\Theta - grande$, inventadas por Paul Bachman, Edmund Landau y otros. A continuación se definen estas dos.

Definición 2.2.7. *($\Omega - grande$.) Un algoritmo tiene complejidad $f(n) = \Omega(g(n))$ si existen las constantes positivas, n_0 y c tales que $\forall \ n \geq n_0$, $f(n) \geq cg(n)$.*

Es decir, $cg(n)$ es una cota inferior de $f(n)$.

Definición 2.2.8. *($\Theta - grande$.) Un algoritmo tiene complejidad $f(n) = \Theta(g(n))$ si existen las constantes positivas, n_0, c_1 y c_2 tales que $\forall n \geq n_0$, se cumple $c_1 g(n) \leq f(n) \leq c_2 g(n)$.*

Significa que $c_1 g(n)$ es una cota inferior para $f(n)$ y $c_2 g(n)$ es una cota superior para $f(n)$, para todo $n \geq n_0$ y esto significa que $g(n)$ proporciona un límite en $f(n)$.

EnLa complejidad $Big - \Theta$ define la cota (superior e inferior) del tiempo de complejidad de un algoritmo.

El análisis asintótico de los algoritmos permite hacer comparaciones entre las tasas promedio de tiempo que emplean los algoritmos en el peor de los casos.

2.2.1. Clases P y NP

La complejidad de un problema es equivalente a la complejidad del algoritmo que lo resuelva. Un problema es *tratable* (o fácil) si existe un algoritmo en tiempo polinomial que lo resuelva. Un problema es *intratable* (o difícil) si no existe un algoritmo en tiempo polinomial que lo resuelva. La teoría de la complejidad de los problemas se encarga de los problemas de decisión. [24]

Para comparar la complejidad de resolución de los problemas se tienen las clases de complejidad computacional. Una clase de complejidad representa el conjunto de problemas que pueden resolverse con determinada cantidad de recursos computacionales. Hay dos clases importantes: $\mathbf{P}$ y $\mathbf{NP}$. En la figura 2.3 se muestra la relación entre estas clases y otra clasifica a los problemas como $\mathbf{NP} - \mathbf{Completos}$ los cuales se definirán más adelante.

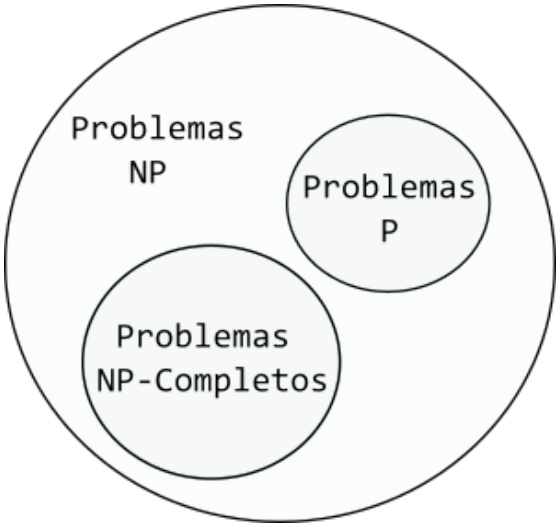


Figura 2.3: Clasificación de los problemas de optimización según su complejidad.

Definición 2.2.9. *(Clase $\mathbf{P}$). Un problema de decisión pertenece a la clase $\mathbf{P}$ (Polinomial-time), si existe un algoritmo que resuelve cualquiera de sus instancias en tiempo polinomial.*

La clase de complejidad $\mathbf{P}$ representa al conjunto de todos los problemas de decisión que pueden ser resueltos por un algoritmo determinista. Por lo tanto, la clase $\mathbf{P}$ representa el conjunto de problemas para los cuales existe un algoritmo polinómico que los resuelve. Y los problemas que pertenecen a esta clase son relativamente "fáciles" de resolver. Algunos de los problemas pertenecientes a esta clase son: el árbol de expansión mínima, el problema de la ruta más corta, el problema de red de flujo máximo, entre otros.

Definición 2.2.10. *(Clase $\mathbf{NP}$). Un problema de decisión pertenece a la clase $\mathbf{NP}$ (Nondeterministic Polinomial) si para cualquiera de sus instancias las soluciones pueden ser verificadas en tiempo polinomial.*

La clase de complejidad $\mathbf{NP}$ corresponde al conjunto de todos los problemas de decisión cuya solución puede ser validada a través de un algoritmo de verificación en tiempo polinomial, un

algoritmo de verificación codifica cualquier solución del problema de decisión cuya respuesta es "sí" y certifica la validez de la respuesta en tiempo polinomial.

Algunos problemas pertenecientes a esta clase son: el problema de la mochila binario, el problema del clique, el problema de cobertura de nodos, el problema del agente viajero, el problema de satisfacibilidad (SAT), entre otros más.

La relación entre los problemas de la clase $\mathbf{P}$ y $\mathbf{NP}$ es que $\mathbf{P} \subseteq \mathbf{NP}$, esto ha desarrollado un estudio en torno a la pregunta ¿$\mathbf{P} = \mathbf{NP}$?.[1] Con esto se tiene el problema $\mathbf{P}$ versus $\mathbf{NP}$, es decir, consiste en determinar si se verifica que $\mathbf{P} \subsetneq \mathbf{NP}$ o que $\mathbf{P} = \mathbf{NP}$. Ya que informalmente e intuitivamente puede decirse que hallar una solución correcta de un problema es estrictamente más difícil que verificar que una solución propuesta del mismo es efectivamente una solución correcta.

Entonces para justificar que $\mathbf{P} \neq \mathbf{NP}$ se utilizan como candidatos los problemas de la clase $\mathbf{NP}$ que no son tratables, es decir, los "más difíciles" mejor conocidos como problemas de la clase $\mathbf{NP - Completos}$, para definirlos tendremos que hablar un poco sobre la reducción entre problemas.

La teoría de reducción fue introducida por Stephen Cook en 1971, en el documento titulado "La Complejidad de los Procedimientos de Demostración de Problemas" (*The Complexity of Theorem Proving Procedures*). [7]

Definición 2.2.11. *(Reducción polinomial en tiempo). Sean* L_1 *y* L_2 *problemas de decisión. Se dice que hay una reducción de tiempo polinomial del problema* L_1 *a* L_2 *si y sólo si el primero puede ser resuelto por un algoritmo* A_1 *que equivale a un número polinomial de llamadas a un algoritmo* A_2 *para resolver el problema* L_2*. [22]*

En consecuencia, si el problema L_1 se reduce de manera polinomial a L_2 y existe un algoritmo polinomial para L_2, entonces también existe un algoritmo de tiempo polinomial para L_1. Una transformación de tiempo polinomial es un caso especial de una reducción de tiempo polinomial que es relevante en el contexto de la teoría de la complejidad.

Definición 2.2.12. *(Transformación en tiempo polinomial). Sean* L_1 *y* L_2 *problemas de decisión. Se dice que existe una tranformación en tiempo polinomial del problema* L_1 *al problema* L_2 *si se puede construir una instancia* J_2 *de* L_2 *en tiempo polinomial a partir de cualquier instancia* J_1 *de* L_1*, de manera que* J_1 *es una instancia cuya respuesta es "sí" de* L_1 *si y solo si* J_2 *es una instancia cuya respuesta es "sí" de* L_2*. [22]*

[1] Esta pregunta es uno de los grandes problemas matemáticos del milenio para el cual, si se resuelve, se da un premio de US$1,000,000 si se sustenta y es aprobada la solución.

Definición 2.2.13. *(Clase* $\mathbf{NP-Completo}$*). Un problema de decisión de la clase* $\mathbf{NP}$ *pertenece a la clase* $\mathbf{NP-Completo}$ *si cualquier otro problema de la clase* $\mathbf{NP}$ *puede ser transformado a él en tiempo polinomial. [22]*

Entonces para demostrar que un problema pertenece a la clase $\mathbf{NP-Completo}$ implican dos pasos principales: primero se tiene que demostrar que es $\mathbf{NP}$, lo segundo es mostrar que todos los demás problemas en $\mathbf{NP}$ pueden ser transformados en tiempo polinomial. Esta segunda parte puede ser la más difícil, ya que se debe mostrar que un problema de la clase $\mathbf{NP-Completo}$ puede ser transformado en tiempo polinomial al problema que queremos. Esto es mencionado en el Teorema de Cook.

Teorema 2.2.1 (Teorema de Cook)**.** *[19] Para probar que un problema pertenece a la clase* $\mathbf{NP-Completos}$*, se debe demostrar que:*

a) El problema pertenece a la clase $\mathbf{NP}$*.*

b) Todos problemas de la clase $\mathbf{NP}$ *se reducen en tiempo polinomial en el problema.*

Para la reducción en tiempo polinomial de un problema a otro se utiliza la propiedad de transitividad a tiempo polinomial se tiene la propiedad de transitividad, la cual dice que si un problema L_1 se reduce polinomialmente al problema L_2, y L_2 se reduce polinomialmente al problema L_3, entonces L_1 se reduce polinomialmente a L_3. Y también se cumple que i un problema L_1 se reduce polinomialmente al problema L_2 entonces L_2 se reduce polinomialmente a L_1.

Definición 2.2.14. *(Clase* $\mathbf{NP-duro}$*). Un problema pertenece a la clase* $\mathbf{NP-duro}$ *si su problema de decisión pertenece a la clase* $\mathbf{NP-Completo}$*, entonces su problema de optimización es* $\mathbf{NP-duro}$*.*

Para los problemas $\mathbf{NP-duros}$ no existen algoritmos eficientes que los resuelvan, ya que requieren un tiempo exponencial (a menos que $\mathbf{P=NP}$) para ser resueltos de manera óptima. Y las *metaheurísticas* son una alternativa importante para resolver esta clase de problemas.

De esta manera, se comienzan a enunciar una serie de teoremas que resultan importantes para la justificación del problema del agente viajero. Estas implicaciones y teoremas pueden consultarse en el libro de Papadimitrou [19] (Capítulo 15, sección 5).

Una vez mencionada la definición de reducción polinomial, se sigue la demostración de pertenencia a la clase NP-Completo. Se demuestra que un problema pertenece a la clase NP, y dado

que el problema del SAT es de la clase NP, existe un algoritmo no determinístico de reducción polinomial que lo transforme entonces, para cualquier problema de la clase NP existe una reducción polinomial a él.

2.2.2. Análisis de la complejidad computacional del PAV

El problema del agente viajero pertenece a la clase $\mathbf{NP}$. [22](Capítulo 2).

Sea una matriz simétrica de nxn con $n \in \mathbb{N}$ de enteros no negativos d_{ij}, y un entero B, averiguar si existe un ciclo hamiltoniano o una permutación cíclica (tour) τ tal que $\sum_{j=1}^{n} d_{i\tau(j)} \leq B$.

Éste problema pertenece a la clase $\mathbf{NP}$ porque para cualquier instancia de tamaño n del PAV cuya respuesta es "sí", podemos demostrar este hecho exhibiendo un ciclo hamiltoniano apropiado. Entonces podría verificarse que para esa instancia el ciclo hamiltoniano τ satisface $\sum_{j=1}^{n} d_{i\tau(j)} \leq B$.

Como se ha mencionado el PAV pertenece a la clase NP y ahora se desarrolla la justificación de que el PAV es NP-Completo. Para realizar la justificación matemática de esta afirmación, se mencionan una serie de definiciones, teoremas y proposiciones. El primer problema a mencionar es el de *satisfacibilidad* o mejor conocido como SAT que será la base del importante teorema de Cook para después concluir que en efecto el PAV es NP-completo.

El problema de satisfacibilidad boolena SAT

Una variable es booleana si solo toma dos valores, el 0 para falso y el 1 para verdadero. Al tomar un conjunto de variables boolenas y asignarles valores se puede analizar si dada una fórmula booleana es cierta o no. Por ejemplo, para las variables $x_1 = 1, x_2 = 1, x_3 = 0$, se deduce que la siguiente fórmula es verdadera.

$$\neg x_3 \wedge (x_1 \vee \neg x_2 \vee x_3) = 1 \wedge (1 \vee 0 \vee 0) = 1$$

Cuando una fórmula booleana es verdadera, se dice que es *satisfacible*. Si se tiene más de una expresión entre paréntesis, es decir, subfórmulas booleanas se le llama *cláusulas*. A continuación se muestra una fórmula con cinco cláusulas booleanas, que resulta ser insatisfacible, ya que no es

verdadera.

$$(x_1 \vee x_2 \vee x_3) \wedge (x_1 \vee \neg x_2) \wedge (x_2 \vee \neg x_3) \wedge (x_3 \vee \neg x_1) \wedge (\neg x_1 \vee \neg x_2 \vee \neg x_3)$$

Observemos que el tamaño del espacio del problema es 2^n, donde n es el número de variables booleanas, debido a que una variable puede tomar dos valores 0 o 1.

Definición 2.2.15 (Problema de satisfacibilidad SAT)**.** *Dadas* m *cláusulas ajenas* $C_1, C_2, \ldots, C_m$*, con las variables booleanas* $x_1, x_2, \cdots, x_n$*. ¿La fórmula booleana dada por* $C_1 \wedge C_2 \wedge \ldots \wedge C_m$ *es satisfacible?*

En otras palabras, el problema del SAT consiste en responder si un sistema dado de fórmulas lógicas en forma conjuntiva tiene solución. Es decir, que al asignarle el valor correspondiente a cada variable y plantear la fórmula, se obtiene como resultado un 1. Como se dedujo, el número de asignaciones posibles es 2^n, por lo que la complejidad de resolver el problema del SAT es deducir en cuantos pasos se logra aceptar si un problema tiene o no solución. Y se dice que la complejidad del problema es NP. Los siguientes teoremas pueden encontrarse en el libro de Papadimitriou [19](Capítulo 15).

Teorema 2.2.2 (SAT)**.** *El problema de satisfacibilidad es NP-Completo.*

Como se mencionó en el teorema de Cook, para demostrar que un problema es NP-Completo deben cumplirse dos puntos. En el primer punto indica que el problema de satisfacibilidad es NP. Por lo tanto, la justificación del segundo punto es que dado cualquier problema de la clase NP se puede transformar en tiempo polinomial en un problema de satisfacibilidad. Para esto se toma cualquier problema, $L \in NP$ y para un conjunto X de variables booleanas se construye una fórmula booleana $F(x)$, de tal forma que X es un ejemplo con resultado 1 de L sí y sólo sí $F(x)$ es satisfacible.

El resultado inmediato que implica este teorema es el problema 3-satisfacibilidad, donde se restringe a tener tres variables en cada cláusula. Que sigue siendo un problema NP, por definición de las variables y se enuncia a continuación.

Teorema 2.2.3. *El problema 3-satisfacibilidad es NP-Completo.*

Este problema es un caso particular del problema de satisfacibilidad. Para la demostración de este, se utiliza una transformación polinómica de satisfacibilidad a 3-satisfacibilidad. Se construye

una fórmula boolena $F'(x)$, en donde solamente se tienen 3 variables por cláusula y se utiliza la anterior $F(x)$, de tal manera que se demuestra que $F'(x)$ es satisfacible si y solo si $F(x)$ lo es. Por la construcción de F' se puede realizar en tiempo polinomial entonces, se dice que existe una transformación en tiempo polinomial de satisfacibilidad a 3-satisfacibilidad. Y se concluye que el problema 3-satisfacibilidad es NP-Completo.

Teorema 2.2.4. *El problema del circuito hamiltoniano es NP-Completo.*

Se sabe que el problema del circuito hamiltoniano pertenece a la clase NP. La demostración de este teorema hace uso del teorema 3-satisfacibilidad mediante una transformación de reducción polinómica a este problema.

Teorema 2.2.5. *El problema del camino hamiltoniano es NP-Completo.*

El problema del circuito hamiltoniano es un problema NP-completo, el cual es un caso particular del PAV. Sea $G = [X, A]$ una gráfica cualquiera, se construye una instancia del PAV con $|X|$ el número de ciudades, sea $d_{ij} = 1$ si $(a_i, a_j) \in A$ y $d_{ij} = 2$ en otro caso. Es inmediato ver que existe un circuito menor o igual que L si y solo si existe un circuito hamiltoniano en G. Por lo tanto la versión de reconocimiento del problema del agente viajero es NP-completo.

Corolario 2.2.5.1. *El Problema del Agente Viajero es NP-Completo.*

Para esta demostración se toma en cuenta que en efecto el problema del circuito hamiltoniano es un caso especial del PAV. Entonces, dada cualquier gráfica $G = [X, A]$, logramos construir una instancia con $|X|$ número de ciudades tal que $d_{ij} = 1$ para $(a_i, a_j) \in A$ y $d_{ij} = 2$ en otro caso.

2.3. Heurísticas y Metaheurísticas

El término de *heurística* proviene de una palabra griega relacionada con el concepto de "encontrar" y de la exclamación *eureka* originaria de Arquímedes al descubrir su famoso principio. Entonces el término heurística [13] se refiere a un procedimiento que encuentra soluciones con un alto grado de confianza de un problema a un costo computacional razonable, aunque no se garantice optimalidad o factibilidad. Una solución *heurística* tendrá un alto grado de optimalidad y/o factibilidad.

El término *metaheurística* [13] se obtiene de anteponer a la palabra *heurística* el prefijo *meta* que significa "más allá" o "a un nivel superior". Este término apareció por primera vez en un artículo sobre búsqueda tabú de Fred Glover en 1986. Las *metaheurísticas* son estrategias inteligentes para diseñar o mejorar procedimientos heurísticos generales con alto rendimiento.

Hay heurísticas para resolver un problema de optimización que son más generales o específicas que otras. Las heurísticas específicas deben ser diseñadas de acuerdo a la información disponible del problema. Y las heurísticas más generales, presentan ventajas como la sencillez, adaptabilidad y robustez de los procedimientos.

2.3.1. Tipos de heurísticas y metaheurísticas

Definición 2.3.1. *Un método heurístico [12] es un procedimiento para resolver un problema de optimización bien definido mediante una aproximación intuitiva, en la que la estructura del problema se utiliza de forma inteligente para obtener una buena solución.*

Los métodos heurísticos pueden ser de naturaleza diferente, por lo que es complicado dar una clasificación completa. A continuación se dan unas categorías amplias, no excluyentes, para ubicar a las heurísticas más conocidas:

- *Métodos de descomposición: el problema original se descompone en subproblemas más sencillos de resolver.*
- *Métodos inductivos: La idea de estos métodos es generalizar de versiones pequeñas o más sencillas al caso completo.*
- *Métodos de reducción: consiste en identificar propiedades que se cumplen mayoritariamente por las buenas soluciones e introducirlas como restricciones al problema.*
- *Métodos constructivos: consisten en construir literalmente paso a paso una solución del problema. Usualmente son métodos deterministas y suelen estar basados en la mejor elección cada iteración.*
- *Método de búsqueda local: los procedimientos de búsqueda local o mejora local comienzan en una solución del problema y lo mejoran progresivamente. El procedimiento realiza en cada paso da una solución a otra con mejor valor.*

Los métodos constructivos y los de búsqueda local constituyen la base de los procedimientos metaheurísticos.

Definición 2.3.2. *Las metaheurísticas [13] son estrategias para diseñar procedimientos heurísticos. Por tanto, los tipos de metaheurísticas se establecen en primer lugar, en función del tipo de procedimientos a los que se refiere.*

Entonces podemos definir los siguientes tipos de metaheurísticas: [13]

- *Las metaheurísticas de relajación: se refieren a procedimientos de resolución de problemas que utilizan relajaciones del modelo original, cuya solución facilita la solución del problema original.* Entre ellas se encuentran los métodos de relajación lagrangiana o de restricciones subordinadas.
- *Las metaheurísticas constructivas: se orientan a los procedimientos de la obtención de una solución a partir del análisis y sección paulatina de los componentes que la conforman.* En este tipo de metaheurísticas se encuentra la estrategia voraz o *greedy*, donde se buscan mejores resultados inmediatos sin tener en cuenta una perspectiva más amplia; y se encuentra la metaheurística conocida como *procedimiento de búsqueda adaptativa aleatoria voraz* (GRASP, por sus siglas en inglés).
- *Las metaheurísticas de búsqueda: guían los procedimientos que usan transformaciones o movimientos para recorrer el espacio de soluciones alternativas y explotar las estructuras asociadas.*
- *Las metaheurísticas evolutivas: están enfocadas a los procedimientos basados en conjuntos de soluciones que evolucionan sobre el espacio de soluciones.* Este tipo de metaheurísticas utilizan subconjuntos del espacio de soluciones factibles, para evolucionarlos mediante mecanismos estratégicos con el fin de construir nuevas soluciones. Aquí se encuentran los algoritmos genéticos, meméticos y las metaheurísticas de búsqueda dispersa o de reencadenamiento de caminos.

Otras metaheurísticas están orientadas en algún tipo de recurso computacional como: las redes neuronales, los sistemas de hormigas o la programación por restricciones.

Capítulo 3

Búsqueda Local

3.1. Introducción a la búsqueda local

La búsqueda local es la base de la construcción de muchos algoritmos y heurísticas. Básicamente es un proceso iterativo en el que se comienza con una solución inicial a la cual se le aplican ciertas "modificaciones" que permiten crear su vecindad.

La búsqueda local en general es un método determinístico y sin memoria, tiene la gran desventaja de quedar atrapado en óptimos locales por lo que, se vuelve importante la construcción de una solución inicial y las vecindades.

Se puede construir la búsqueda local mediante el siguiente pseudocódigo:

Algoritmo 1: Búsqueda local

Inicialización
s = solución inicial
Mientras (s no es óptimo local):
Generar $s' \in N(s)$
Si c(s') <c(s)
(la solución vecina s' es mejor que s)
$s \leftarrow s'$
fin
retornar s

Anteriormente se explicó que la *vecindad* de una solución son todas las posibles soluciones generadas a partir de movimientos definidos en un algoritmo. Entonces la definición de los movimientos dependen del problema y de su función objetivo. La manera de seleccionar una solución puede ser definida bajo distintos criterios, uno de los más utilizados es elegir una nueva solución que sea mejor que la actual, es decir, que tenga una mejor evaluación en la función objetivo.

La búsqueda se detiene cuando el valor de la función objetivo no puede ser mejorado. A la solución encontrada se le denomina *óptimo local* respecto a la vecindad definida.

La búsqueda local no garantiza alcanzar el óptimo global para el problema, ya que por la "miopía" del procedimiento, para problemas con cierta dificultad será poco garantizado obtenerlo.

Entonces pueden establecerse criterios de paro de acuerdo al comportamiento de la búsqueda, ya sea utilizando un número de iteraciones máximas, o si existe la posibilidad de quedarse atrapado en un óptimo local entonces establecer un criterio para cambiar a otra solución e intentar seguir explorando más vecindades, entre otros criterios que permitan buscar en el espacio de búsqueda.

Algunas construcciones de búsqueda local son:

Búsqueda local rápida. La idea de esta construcción es ignorar los vecinos que no mejoren la calidad de la solución, asignando un bit de activación con valor de 1 si se observa que los movimientos aplicados a la solución inicial produce una vecindad con mejores costos en las soluciones, en caso contrario se desactiva el valor del bit.

Búsqueda de reinicio aleatorio y multi-inicio. La forma de hacer esta búsqueda es comenzar en distintos puntos aleatorios, con el fin de lograr diversidad y así no aterrizar de manera pronta en óptimos locales. Una desventaja es que al incrementar el tamaño del espacio de bús-

queda del problema esta estrategia se vuelve menos efectiva. Esta búsqueda se compone de dos fases principales: en la primera se construye una solución inicial y en la segunda se aplica una mejora a la solución obtenida de la primera fase.

Búsqueda local guiada. En esta búsqueda se modifica la función objetivo mediante ciertas penalizaciones que permiten un mecanismo de mayor exploración.

También existen mecanismos de búsqueda local en los que se aceptan vecinos de no mejora a la solución inicial, con la esperanza de permitir un movimiento tal que se explore el espacio de soluciones y se encuentre un nuevo óptimo local. Dentro de esta categoría se encuentran: búsqueda tabú y recocido simulado.

3.2. Búsqueda local para el PAV

En el presente trabajo la búsqueda local del PAV se realiza mediante el intercambio de dos ciudades a partir de una solución inicial la cual se construye con la heurística del vecino más cercano.

Las vecindades del problema están definidas por los movimientos o intercambios de ciudades o aristas, mejor conocidos como movimientos *k - Opt* donde de manera general se eligen un número k de aristas o vértices de la solución actual y se genera una nueva solución reordenando las k aristas o intercambiando las k ciudades. Este tipo de movimientos conforman la base de las heurísticas de búsqueda local más usadas para el PAV como: 2-Opt, 3-Opt y Lin-Kernighan (LK).

3.2.1. Heurística del vecino más cercano

La heurística del vecino más cercano es quizá una de las más simples y sencillas que se utilizan para construir una solución inicial del PAV. Rosenkrantz, Stearns y Lewis en 1977 dieron origen a esta heurística. [12]

La clave es que a partir de una ciudad i en cada iteración se elige la ciudad j más cercana con el fin de formar un ciclo hamiltoniano de menor distancia. Debido a su procedimiento de resolución se cataloga como una heurística constructiva. A continuación se muestra su pseudocódigo.

Algoritmo 2: Vecino más Cercano

Inicialización

Seleccionar un vértice $i \in V$ al azar.

Hacer W $= V \backslash \{i\}$.

$s = \{i\}$

Mientras ($W \neq \varnothing$)

Seleccionar $j \in W$ tal que $d = \min\{d_{ij}$ para $j \in W\}$

Añadir $s = s \cup \{j\}$

Actualizar $W = W \setminus \{j\}$

$i = j$.

Añadir ciudad inicial $s = s \cup \{i\}$

FIN

En otras palabras, la serie de pasos a seguirse son:

1. Seleccionar aleatoriamente un vértice ciudad como punto de partida.
2. Ir al vértice vecino no visitado más cercano o de menor distancia del último vértice considerado e incluirlo en el tour (rompiendo empates arbitrariamente).
3. Si aún existen vértices no visitados, ir al paso 2.
4. Finalmente, vinculamos el último vértice del tour con el primero, obteniendo el ciclo hamiltoniano.

Ejemplo

Sea la siguiente matriz de distancias en kilómetros de un modelo PAV simétrico con 5 ciudades.

$$\|d_{ij}\| = \begin{pmatrix} \infty & 120 & 220 & 150 & 210 \\ 120 & \infty & 100 & 110 & 130 \\ 220 & 100 & \infty & 160 & 185 \\ 150 & 110 & 160 & \infty & 190 \\ 210 & 130 & 185 & 190 & \infty \end{pmatrix}$$

Se puede iniciar en cualquier ciudad, dependiendo de esto la distancia de la solución puede variar. Se selecciona iniciar en la ciudad 3.

Iteración 1. Iniciar en la ciudad 3.

Se añade la ciudad inicial $s = \{3\}$ y se tiene el conjunto $W = \{1, 2, 4, 5\}$.

Iteración 2. De las ciudades adyacentes a la ciudad 3, la ciudad 2 es la más cercana a la ciudad 3 con 100 unidades.

$$\begin{aligned} \min &= \{d_{31}, d_{32}, d_{33}, d_{34}, d_{35}\} \\ &= \{220, 100, \infty, 160, 185\} \\ &= 100 \end{aligned}$$

Actualizando: $s = \{3, 2\}$, $W = \{1, 4, 5\}$.

Iteración 3. De las ciudades no visitadas y adyacentes a la ciudad 2, elegir la más cercana, la ciudad 4 con 110 unidades.

$$\begin{aligned} \min &= \{d_{21}, d_{22}, d_{23}, d_{24}, d_{25}\} \\ &= \{120, \infty, -, 110, 130\} \\ &= 110 \end{aligned}$$

Actualizando: $s = \{3, 2, 4\}$, $W = \{1, 5\}$

Iteración 4. De las ciudades no visitadas y adyacentes a la ciudad 4, elegir la más cercana, la ciudad 1 con 150 unidades.

$$\begin{aligned} \min &= \{d_{41}, d_{42}, d_{43}, d_{44}, d_{45}\} \\ &= \{150, -, -, \infty, 190\} \\ &= 150 \end{aligned}$$

Actualizando: $s = \{3, 2, 4, 1\},\ W = \{5\}$

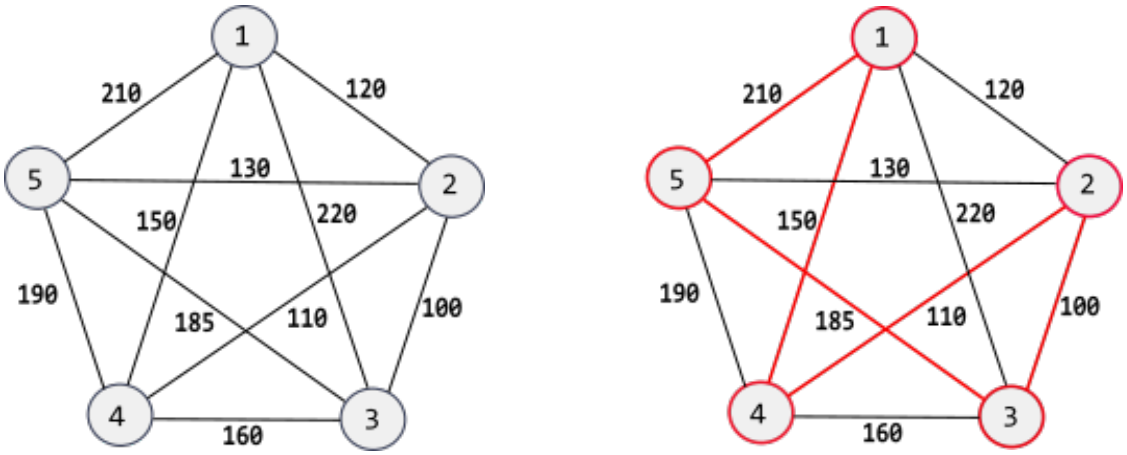
Iteración 5. De las ciudades no visitadas y adyacentes a la ciudad 1 se tiene la ciudad 5 con 210 unidades.

$$\begin{aligned} \min &= \{d_{51}, d_{52}, d_{53}, d_{54}, d_{55}\} \\ &= \{\infty, -, -, -, 210\} \\ &= 210 \end{aligned}$$

Actualizando: $s = \{3, 2, 4, 1, 5\},\ W = \{\varnothing\}$

Iteración 6. Agregar la ciudad 3 para completar el ciclo hamiltoniano.

Solución final: $s = \{3, 2, 4, 1, 5, 3\}$



(a) Gráfica completa con 5 ciudades y la distancia entre cada una de ellas.

(b) Ciclo hamiltoniano resultante: 3->2->4->1->5->3.

Figura 3.1: Gráfica de 5 ciudades y una solución generada por la heurística del vecino más cercano

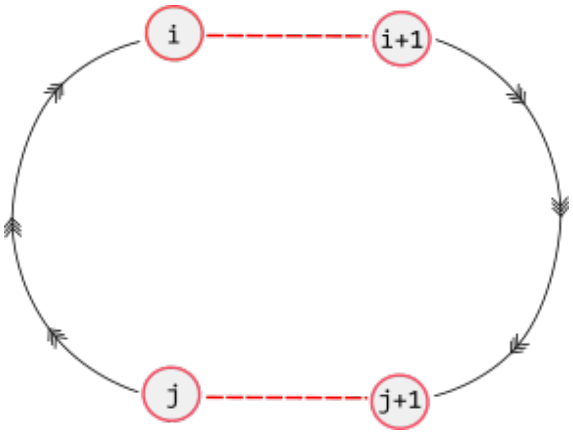
Se aprecia que para ejemplos con un número pequeño de ciudades, es fácil realizar la comparación de aristas de menor distancia. Además, se sabe que el número de ciclos hamiltonianos posibles en una gráfica de 5 ciudades son $\frac{(5-1)!}{2} = 12$ posibles soluciones a las cuales se les asocia su respectivo costo de función objetivo y se comparan para elegir el ciclo de menor distancia, para una gráfica de 10 ciudades el número de ciclos hamiltonianos posibles son 181,440, por lo

que se convierte en un camino no viable para encontrar la mejor solución. Y con la estrategia de selección de aristas al elegir la mejor opción en cada paso, sin ver que esto puede obligarnos a tomar malas decisiones en iteraciones posteriores, nos indica que seleccionar un algoritmo miope y determinístico no basta para agilizar la búsqueda de la mejor solución.

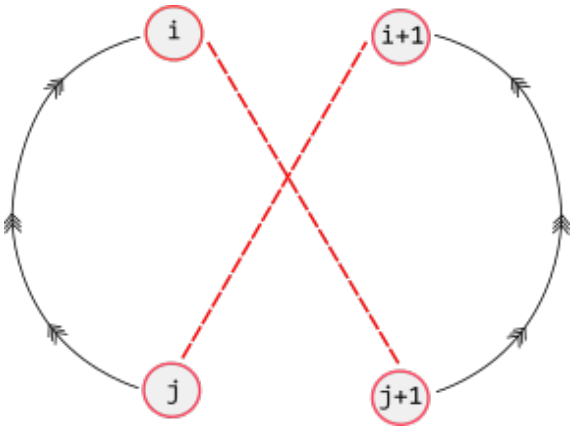
3.2.2. 2-intercambio

El procedimiento de vecinos 2-intercambio está inspirado en los movimientos k-intercambios. Un *movimiento* 2-intercambio consiste en eliminar dos aristas y reconectar los dos caminos de una manera diferente para obtener un nuevo ciclo [12].

Entonces lo que se realiza es un *intercambio* de dos vértices seleccionados a partir de una solución inicial tal que se obtiene un nuevo ciclo hamiltoniano con un costo igual o diferente. Gráficamente se explica en la figura 3.2



(a) Seleccionar aleatoriamente 2 vértices i, j, se toman las aristas (i,i+1) y (j,j+1) las eliminamos del ciclo.



(b) Acomodar el vértice i en la posición de j y viceversa, de tal forma que se tengan las nuevas aristas $(i, j+1)$ y $(j, i+1)$, este es el nuevo ciclo.

Figura 3.2: Ejemplo del 2-intercambio.

El conjunto de soluciones vecinas de una solución inicial son todas aquellas resultantes después de hacer un intercambio de posición de dos ciudades i, j tal que $1 \leq i < j \leq n$ originando nuevos ciclos hamiltonianos.

Comprobar si existe, o no, un movimiento 2-intercambio examinando todos los pares de aristas

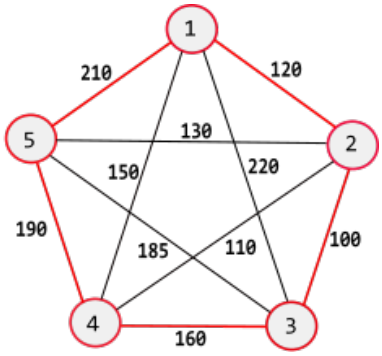
en el ciclo, significa revisar

$$\frac{n(n-1)}{2}$$

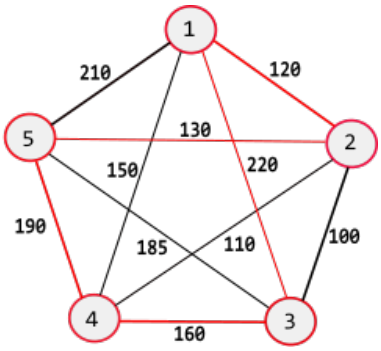
posibles soluciones, ya que son las combinaciones $\binom{n}{2}$ que representa el número de subconjuntos de 2 ciudades del conjunto total de n ciudades. Por lo tanto se ocupa un tiempo de orden $\mathcal{O}(n^2)$.

De la gráfica del ejemplo 3.1a, se da una solución inicial la cual corresponde a la figura 3.3a con una distancia inicial de 780 unidades. Las soluciones vecinas generadas son las gráficas que se muestran en la figura 3.3.

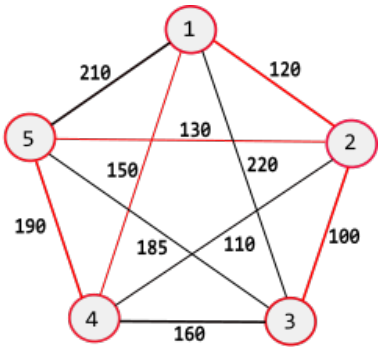
En la figura 3.3a se observa que la gráfica tiene $\binom{5}{2} = 10$ soluciones vecinas, la mejor solución vecina tiene una distancia de 745 unidades, la cual mejora el valor de la distancia inicial. Igualmente se observa que no necesariamente las soluciones vecinas tienen un mejor costo al inicial.



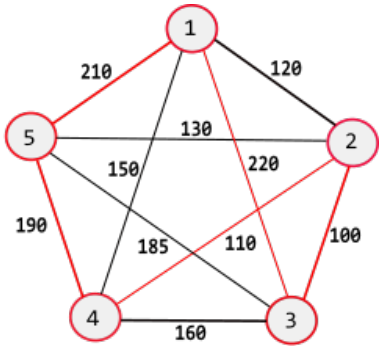
(a) Solución inicial, $c(s_0) = 780$



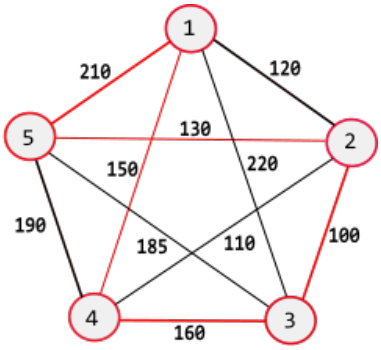
(b) $i = 1, j = 2$
$c(s_1) = 820$



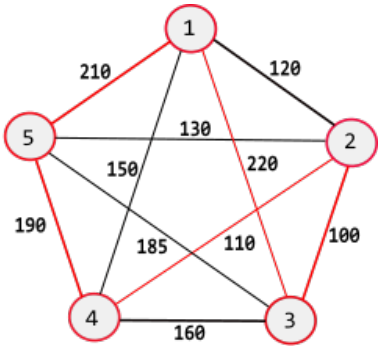
(c) $i = 1, j = 3$
$c(s_2) = 745$



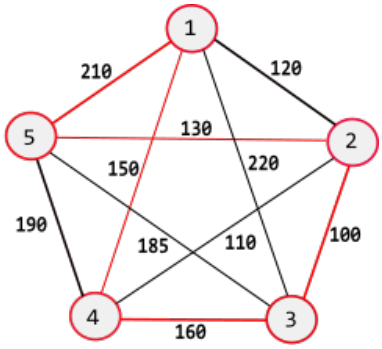
(d) $i = 1, j = 4$
$c(s_3') = 830$



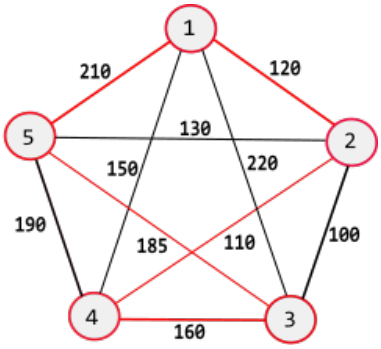
(e) $i = 1, j = 5$
$c(s_4) = 750$



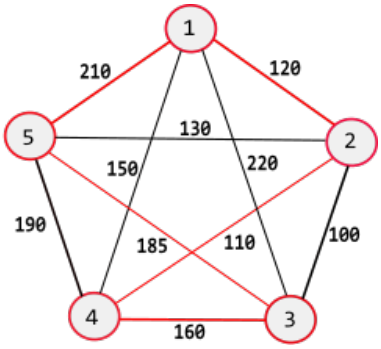
(f) $i = 2, j = 3$
$c(s_5) = 830$



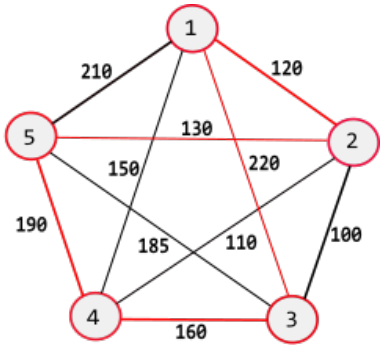
(g) $i = 2, j = 4$
$c(s_6) = 750$



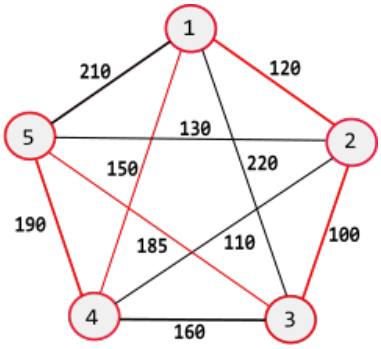
(h) $i = 2, j = 5$
$c(s_7) = 785$



(i) $i = 3, j = 4$
$c(s_8) = 785$



(j) $i = 3, j = 5$
$c(s_9) = 820$



(k) $i = 4, j = 5$
$c(s_{10}) = 745$

Figura 3.3: Soluciones vecinas creadas a partir del intercambio de las dos ciudades i, j utilizando como solución inicial el inciso (a). Generadas en *Inkscape*. Fuente: [22] (pp.81,82)

3.2.3. Construcción una búsqueda local

Una vez presentados los dos algoritmos que se utilizarán para generar un espacio de búsqueda, se explica la estrategia que se usa para aplicar búsqueda local al problema. De manera sencilla se realiza lo siguiente:

i) Generar una solución inicial a partir del vecino más cercano, iniciando en una ciudad establecida.

ii) A partir de la solución inicial aplicar la búsqueda local 2-intercambio para generar una solución vecina.

iii) Si el valor de una solución vecina es mejor, entonces se reemplaza como la mejor.

iv) Repetir ii) y iii) hasta que se cumpla el criterio de paro, en este caso se utiliza el número de iteraciones.

Capítulo 4

Recocido Simulado

4.1. Antecedentes del recocido simulado

Las primeras ideas de este algoritmo iniciaron con Nicholas Metropolis [14] las cuales se encuentran en su artículo publicado *Equation of State Calculations by Fast Computing Machines* en el año 1953, donde explica este método que simula el enfriamiento de cierto material en un proceso con calor conocido como *recocido*. Si el material sólido se calienta más allá de su punto de fusión y luego se enfría de nuevo a un estado sólido, las propiedades estructurales dependerán de la velocidad de enfriamiento. Un ejemplo del recocido es la realización de la expansión del cristal en donde debe realizarse un enfriamiento muy lento, ya que si se realiza de manera errónea, el cristal puede tener algún defecto.

Este proceso de recocido puede ser simulado considerando el material como un sistema de partículas. De esta, el algoritmo de *Metropolis* simula el cambio de energía del sistema cuando se somete a un proceso de enfriamiento hasta converger a un estado estable "congelado".

Treinta años después, Kirkpatrick a principios de la década de 1980 dio introducción a los conceptos de recocido para la búsqueda de soluciones factibles para problemas de optimización combinatoria con el objetivo de lograr la convergencia de las soluciones y alcanzar el óptimo local o global.

En física de la materia condensada, el recocido se conoce como un proceso térmico para obtener estados de baja energía de un sólido en un baño de calor. El proceso consta de lo siguiente dos pasos (Kirkpatrick et al. 1983):

- Aumentar la temperatura del baño de calor a un valor máximo en el que el sólido se derrite.
- Disminuir cuidadosamente la temperatura del baño térmico hasta que las partículas se acomoden en el estado fundamental del sólido.

Kirkpatrick menciona [6] que el proceso de recocido es conocido como un proceso térmico utilizado para la disminución de la energía de un sólido en un baño de calor. Y se tienen los siguientes dos pasos:

- Aumentar la temperatura del baño de calor a un valor máximo en el que el sólido se derrite.
- Disminuir cuidadosamente la temperatura del baño de calor hasta que las partículas se coloquen en el estado fundamental del sólido.

Es importante mencionar que el recocido simulado no es propiamente un algoritmo sino una estrategia heurística que necesita de diversas decisiones para su implementación las cuales se ven reflejadas en la calidad de las soluciones resultantes [9]. Como se ha dicho en los capítulos anteriores, la estructura del espacio de búsqueda, la función de costo y la construcción de las vecindades juegan un papel muy importante para explorar de buena manera el espacio de búsqueda y encontrar buenas soluciones.

4.2. Heurística recocido simulado

Los algoritmos de búsqueda local inician con una solución inicial y buscan mejorar la calidad de las soluciones mediante pequeñas perturbaciones como el algoritmo 2-opciones.

Un algoritmo de búsqueda local suele terminar en algún óptimo local, ya que este tipo de algoritmos solo aceptan transiciones correspondientes a la disminución en el costo de la función objetivo. Una manera de no caer en óptimos locales rápidamente se puede lograr permitiendo que algunos movimientos generen soluciones no tan buenas. En el recocido simulado se habla sobre movimientos de escape los cuales son controlados mediante una función de probabilidad, la cual se encarga de no dirigirse hacía peores soluciones.

El recocido simulado está sustentado con el algoritmo conocido como *Metropolis* el cual modela el proceso de recocido, éste simula los cambios energéticos en un sistema de partículas, disminuyendo gradualmente la temperatura hasta alcanzar un estado estable. Las leyes de la termodinámica dicen que a una temperatura t la probabilidad de un incremento energético de magnitud δE se aproxima mediante la siguiente ecuación:

$$\mathbf{P}[\delta E] = exp(-\frac{\delta E}{kt}) \tag{4.1}$$

donde k es una constante conocida como la constante de Boltzmann ($k = 1.38x10^{-23}$ joule/Kelvin) y t es un parámetro de control de temperatura que corresponde a la analogía del proceso del recocido. En el algoritmo de Metropolis se genera una perturbación aleatoria en el sistema y se calculan los cambios de energía resultantes: si se presenta una caída energética el cambio se acepta automáticamente; en caso contrario si se produce un incremento energético este cambio será aceptado con cierta probabilidad la cual se calcula como en la expresión 4.1. El proceso itera hasta cumplir el criterio de paro establecido con el objetivo de que al decrecer la temperatura el sistema logre establecerse.

Tanto Kirkpatrick et al. (1983) y Cerny (1985), dieron la adaptación y analogía de este proceso a la aplicación de problemas de optimización, asociando la equivalencia entre los conceptos teóricos definidos, tal como se muestra en la tabla 4.1:

Simulación termodinámica	**Optimización combinatoria**
Estados del sistema	Soluciones factibles
Energía	Costo
Cambio de estado	Solución del entorno
Temperatura	Parámetro de control
Estado congelado	Solución heurística

Tabla 4.1: Analogía de conceptos entre el proceso y la heurística. Fuente: *Heuristic design and fundamentals of the Simulated Annealing*[9].

4.2.1. Implementación del Recocido Simulado

Para resolver un problema de optimización combinatoria con la heurística del recocido simulado se deben considerar una serie de decisiones las cuales se pueden dividir de la siguiente forma: decisiones genéricas las cuales se toman en cuenta para la implementación general del RS y son conocidas como *esquema de recocido o enfriamiento*; y decisiones específicas que serán particulares para el problema. Estas se presentan en la tabla 4.2.

El buen diseño del recocido simulado pide un buen uso de todos los conocimientos disponibles (teóricos y prácticos) del problema real y la experimentación con los parámetros para encontrar un buen ajuste. A continuación se muestra en la tabla 4.2 los parámetros que son más generales dado que es el esquema del recocido simulado y cuáles son específicos de acuerdo al problema a trabajar.

Genéricas (Esquema de Recocido)	**Problemas Específicos**
t_0 (temperatura inicial) nrep (número de iteraciones) t (función de temperatura) Criterio de Paro	S_0 (solución inicial) Generación de Vecinos Evaluación del δ

Tabla 4.2: Parámetros generales y específicos del recocido simulado[9].

La constante de Boltzmann k, de la ecuación 4.1 en general no es considerada para los problemas de optimización debido a que no tiene algún significado en estos, por lo que se le asigna el valor de 1.

El valor de la temperatura (t) elegido para el algoritmo influirá en los resultados, para el caso de un problema de minimización, cuando t comience a tener valores pequeños, el movimiento a soluciones peores disminuirá y comenzaremos acercarnos cada vez más a un óptimo local.

Por tanto, si una solución candidata tiene un mejor valor en la función objetivo, esta siempre será aceptada. En otro caso, si la solución candidata fuera "peor", la probabilidad de aceptarla dependerá de qué tan peor es y del tamaño de la temperatura. Para eso se habla de una regla de aprobación de movimiento, la cual aceptará un paso si es un poco descendente, pero será poco probable que se acepte si el paso es muy descendente.

Así que la elección de los valores que tome la temperatura en el transcurso del tiempo controla

un grado de aleatoriedad que permite pasos ascendentes. La forma de decidir si el movimiento en forma ascendente será aceptado, será comparando un número aleatorio generado entre 0 y 1 con la probabilidad de aceptación.

$$\begin{gathered}\text{Si número aleatorio} < \text{ Probabilidad (aceptación)} \\ \Longrightarrow \text{aceptamos la solución con un valor ascendente} \\ \text{en otro caso, se rechaza la solución}\end{gathered}$$

En la estructura del algoritmo la cual se escribe al final del capítulo, en el pseudocódigo 3, el enfriamiento debe constar de los parámetros: temperatura inicial, una función que disminuya la temperatura y una condición de paro. En este caso se tiene una forma de enfriamiento geométrica, $t_{k+1} = \alpha t_k$, la cual es una de las más comunes. De acuerdo al artículo [25], los valores más comunes para el parámetro de enfriamiento α que permiten la velocidad de enfriamiento y dan temperaturas moderadamente bajas son de 0.8 a 0.99.

4.3. El recocido simulado para el PAV

Para esta heurística se comienza con una solución inicial y a través del intercambio 2-opciones se generan soluciones vecinas, de esta manera se explora el espacio de soluciones. El recocido simulado se implementa como una estrategia con búsqueda voraz la cual ayuda acelerar la tasa de convergencia para instancias de gran tamaño.

Para ejemplificar el proceso del recocido simulado, se toma la instancia 3.1a de los problemas denominados benchmark.

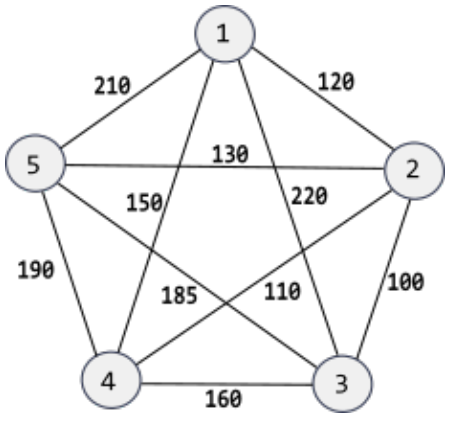


Figura 4.1: Gráfica completa con 5 ciudades y la distancia entre cada una de ellas.

Inicialización de los parámetros:

- $T_0 = 1000$ (temperatura inicial)
- nrep $= 2$ (número de iteraciones por temperatura)
- $\alpha = 0.6$ (factor de reducción de la temperatura)
- $T_f = 0.001$ (temperatura final)
- Criterio de paro: cuando delta $\delta = f(s) - f(s_0)$ sea un valor muy cercano a cero o se alcance la temperatura final.

Se inicia con la siguiente solución inicial y su costo asociado:

$$s_0 = (5,3,1,4,2) \qquad f(s_0) = d(5,3) + d(3,1) + d(1,4) + d(4,2) + d(2,5) = 795$$

Iteración 1. Generar una solución vecina $s \in N(s_0)$, se realiza el intercambio aleatorio de dos vértices (4,2) que genera un intercambio de aristas.

$$s = (5,3,1,2,4) \qquad c(s) = d(5,3) + d(3,1) + d(1,2) + d(2,4) + d(4,5) = 825$$

Calculando el valor de $\delta = f(s) - f(s_0) = 825 - 795 = 30 > 0$ ésto indica que no hay mejora en la calidad de la solución, por lo tanto se acepta con probabilidad $u \in U(0,1) = 0.34 < exp(-\frac{30}{1000}) = 0.97$. Dado que se cumple la desigualdad, se acepta el cambio de la solución y se hace $s_0 = s$ y se avanza a la siguiente iteración.

Iteración 2. Se genera una solución vecina de s_0, mediante el intercambio de dos vértices seleccionados de manera aleatoria, sean estos (5,3) de esta manera se obtiene la solución:

$$s = (3,5,1,2,4) \qquad (s) = d(3,5) + d(5,1) + d(1,2) + d(2,4) + d(4,3) = 785$$

Calculando el valor de $\delta = f(s) - f(s_0) = 785 - 825 < 0$ dado que hay mejora en el valor de la solución, se hace $s_0 = s$.

Se estableció que por cada temperatura se realizan 2 repeticiones por lo tanto se actualiza la temperatura:

$$T = T * \alpha = 1000(0.6) = 600$$

Iteración 1. Generar una solución vecina de s_0, intercambiando aleatoriamente dos vértices, (3,4) se genera la solución:

$$s = (4,5,1,2,3) \qquad f(s) = d(4,5) + d(5,1) + d(1,2) + d(2,3) + d(3,4) = 780$$

Sea $\delta = f(s) - f(s_0) = 780 - 785 = -5 < 0$ dado que mejora el valor de la solución. Se hace $s_0 = s$.

Iteración 2. Generar una solución vecina de s_0, intercambiando aleatoriamente dos vértices, (5,2) se obtiene:

$$s = (4,2,1,5,3) \qquad f(s) = d(4,2) + d(2,1) + d(1,5) + d(5,3) + d(3,4) = 785$$

Sea $\delta = f(s) - f(s_0) = 785 - 780 > 0$ no hay mejora en la solución, se genera un número $u \in U(0,1) = 0.65$ ahora se compara con el valor resultante de $exp(-\frac{\delta}{T}) = exp(-\frac{-785}{600}) = 0.27$.

$$u = 0.65 \leq exp(-\frac{\delta}{T}) = 0.27$$

Dado que no se cumple la desigualdad, no se acepta el cambio de la solución.

Actualización del valor de la temperatura:

$$T = T * \alpha = 600(0.6) = 360$$

Iteración 1. Generar una solución vecina de s_0, intercambiando aleatoriamente dos vértices, (5,1) se genera la solución:

$$s = (4,1,5,2,3) \qquad f(s) = d(4,1) + d(1,5) + d(5,2) + d(2,3) + d(3,4) = 750$$

Sea $\delta = f(s) - f(s_0) = 750 - 780 = -30 < 0$ dado que mejora el valor de la solución. Se hace $s_0 = s$.

Iteración 2. Generar una solución vecina de s_0, intercambiando aleatoriamente dos vértices, (5,2) se genera la solución:

$$s = (4,1,2,5,3) \qquad f(s) = d(4,1) + d(1,2) + d(2,5) + d(5,3) + d(3,4) = 745$$

Sea $\delta = f(s) - f(s_0) = 745 - 750 = -5 < 0$ dado que mejora el valor de la solución. Se hace $s_0 = s$.

Actualización del valor de la temperatura:

$$T = T * \alpha = 360(0.6) = 216$$

Iteración 1. Generar una solución vecina de s_0, intercambiando aleatoriamente dos vértices, (4,2) se genera la solución:

$$s = (2, 1, 4, 5, 3) \qquad f(s) = d(2, 1) + d(1, 4) + d(4, 5) + d(5, 3) + d(3, 2) = 745$$

Sea $\delta = f(s) - f(s_0) = 745 - 745 = 0$ no se mejora ni se empeora el valor de la solución.

Hasta aquí se observa que hay dos soluciones con el mismo valor $f(s)$. Como uno de los criterios de paro establecidos se cumple, ya que el valor de delta tiene un valor de cero entonces el proceso se detiene y se devuelve la última solución.

Por otra parte, si se continúa hasta cumplir el criterio de paro, es decir, alcanzar el valor de la temperatura final entonces se aprecia que el valor obtenido de 745 se encuentra repetido en diversas ocasiones por lo tanto, se dice que las soluciones convergen a este valor.

Algoritmo 3: Recocido simulado

Inicialización de parámetros

Generar una solución inicial s_0

Seleccionar una temperatura inicial $t_0 > 0$

Seleccionar una función de reducción de la temperatura α

Establecer un número de iteraciones *nrep*

Establecer *criterio de paro*

REPETIR

REPETIR

Generar aleatoriamente $s \in N(S_0)$

Sea $\delta = f(s) - f(s_0)$

* Si $\delta \leq 0$ entonces $s_0 = s$

* Si $\delta > 0$, entonces genera aleatoriamente $u \in U(0, 1)$

Si $u < exp(-\delta/t)$ entonces $s_0 = s$

HASTA QUE número_de_iteraciones $=$ *nrep*

$t = \alpha t$

HASTA QUE criterio_de_paro $=$ *VERDADERO*

Nota: La mejor solución visitada será la solución heurística dada por el algoritmo.

Capítulo 5

Algoritmos Genéticos

Los algoritmos genéticos han sido inspirados en la selección natural y la evolución. A partir de un conjunto de soluciones, llamado población inicial, se *evoluciona* explorando el espacio de búsqueda mediante la aplicación de operadores genéticos de selección, recombinación y mutación, para así crear distintas soluciones conocidas como individuos de la población. Todo este proceso de generar individuos en nuevas poblaciones tiene como fin llevar a los mejores calificados para obtener una solución global óptima.

5.1. Antecedentes de los algoritmos genéticos

Diversos investigadores han estudiado la replicación y seguimiento de algunos comportamientos biológicos, que han servido de motivación para algunas estrategias como la programación evolutiva y algoritmos genéticos. Los **algoritmos genéticos** fueron desarrollados por John Holland en 1960 [18] en colaboración con sus estudiantes y colegas de la Universidad de Michigan entre los años *1960's* y *1970's*.

En la teoría biológica de la evolución, los individuos de una población y la adaptación en relación con su entorno juega un papel importante porque se mide la capacidad de supervivencia para poder reproducirse y dejar descendencia para la siguiente generación. A través de cruces entre los individuos de la población se busca obtener mejores valores de aptitud en las generaciones resultantes.

5.2. Terminología para los algoritmos genéticos

A continuación se explican las analogías correspondientes a los términos biológicos originales y la adaptación a la heurística de algoritmos genéticos.

Todos los organismos vivos están compuestos de células que contienen cromosomas dentro de su núcleo. Conceptualmente cada cromosoma puede ser dividido en genes los cuales codifican un rasgo en el individuo, por ejemplo el color de ojos.

En la reproducción sexual hay un proceso de recombinación o cruzamiento de cromosomas de los progenitores a sus descendientes para que exista una variabilidad genética entre los individuos. También se usa el suceso biológico conocido como *mutación* el cual genera cambios en uno o varios genes provocando una variación en la secuencia del ADN. La *aptitud* se define como la probabilidad de que un organismo sea capaz de vivir hasta reproducirse.

Y también se da la analogía para algunos términos biológicos:

- **Población:** es el conjunto de individuos o soluciones construidas.
- **Individuo:** es la unidad de una población, es decir, una solución es considerada como un individuo y cada uno tiene asociado un cromosoma.

- **Cromosoma:** cada cromosoma está representado por un conjunto de valores que definen a la solución. La unidad más pequeña de un cromosoma es llamado *alelo* y se representa con 0 o 1 (bit), como se muestra en la figura 5.1.
- **Gen:** cada cromosoma tiene un conjunto de genes. Cada gen representa los valores de cada variable que conforma a la solución, por ejemplo, para un problema con variables binarias cada una de estas pueden tomar valores de 0 y 1.

Para esta heurística se comienza construyendo una población inicial constituida por soluciones a las cuales se les aplica un proceso o función de *selección natural* mediante operadores bioinspirados como *crossover o recombinación*, *mutación* y *selección*.

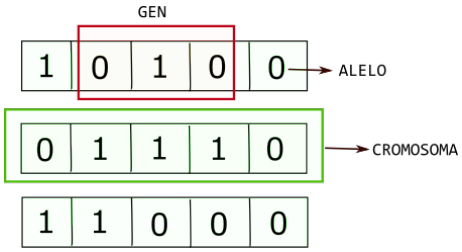


Figura 5.1: Diagrama de una población constituida por 3 cromosomas. Generado en *Inkscape*.

A cada individuo de la población se le asigna un valor de evaluación de acuerdo a una función de aptitud. Esta función aptitud asocia a cada individuo un valor de aptitud física con el propósito de seleccionar los mejores individuos, entre mayor es el valor de la aptitud mejora la calidad de la solución. La selección de los individuos con mejor valor de aptitud, es aplicado para generar un grupo de individuos de apareamiento donde el individuo con mejor calidad o valor de aptitud tiene una mayor probabilidad de ser seleccionado para aparearse.

De un grupo de apareamiento se seleccionan pares de padres y se busca que el entrelazamiento de ellos genere dos descendientes con mejor valor de aptitud, esperando obtener una mejor *calidad* en los descendientes de la generación. La finalidad de mejorar la calidad de las soluciones generadas en cada generación es la aproximación más cercana al valor global óptimo.

5.3. Operadores de los algoritmos genéticos

El objetivo de aplicar operadores genéticos a los individuos o soluciones es tener una mayor diversidad genética en las siguientes generaciones. A continuación se describen los tres tipos de operadores principales los cuales se detallan más adelante:

Selección

En esta fase se seleccionan cierto número de individuos los cuales obtuvieron un valor de aptitud que los cataloga para ser capaces de reproducirse. Estos individuos serán llamados *padres* y con ellos se realiza el entrelazamiento de cromosomas. En la figura 5.2 se tiene que de una población conformada por cuatro soluciones se seleccionan dos candidatas para ser progenitores.

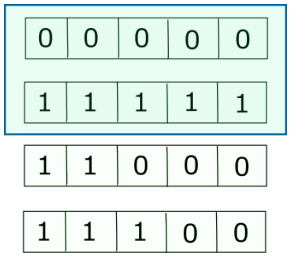
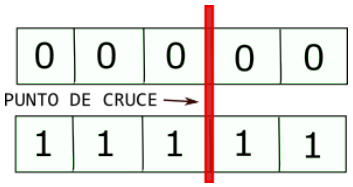


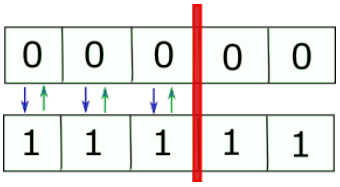
Figura 5.2: Población con 4 individuos de las cuales se eligen las dos primeras para realizar el proceso de recombinación de sus cromosomas. Generado en *Inkscape*.

Cruce o recombinación

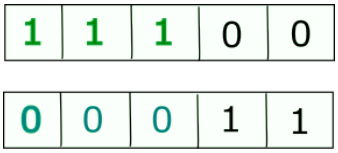
El cruce es una fase en el algoritmo genético de suma importancia porque en ella ocurre la recombinación, es decir, la reproducción de los individuos, esto es, por cada par de padres que se tiene se crean nuevos individuos que tienen un segmento del cromosoma de cada padre. La elección de los segmentos heredados son definidos por los puntos de cruce y elegidos aleatoriamente. Un ejemplo de esto se muestra en la figura 5.3.



(a) De los padres seleccionados ahora se elige aleatoriamente el punto de cruce en cada uno de los cromosomas.



(b) Movimiento de intercambio de los genes para generar los dos descendientes.



(c) Hijos obtenidos de la recombinación de genes.

Figura 5.3: Proceso de crossover o recombinación de los cromosomas.

Mutación

Una vez estructurados los cromosomas de los descendientes se puede aplicar una *mutación o cambio* a los genes con cierta probabilidad. La mutación es un cambio en los cromosomas que tiene como objetivo mantener la variación en la población. En la figura 5.4 se muestra la mutación aplicada a un descendiente.

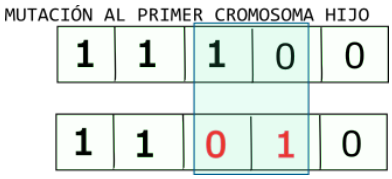


Figura 5.4: De la figura 5.3 se elige el hijo 1 para aplicarle una modificación de los genes 3 y 4 asignándoles el valor contrario al que tenían.

El conjunto de todos los individuos que conforman una nueva población generada es llamada *generación*. La nueva población que se construye suple a la antigua y a este proceso se le llama *reemplazo*.

Lo que sigue es detallar la manera en que se pueden llevar a cabo estos operadores.

5.3.1. Operadores de selección

Selección por ruleta (*roulette-wheel selection*)

El método de la ruleta es un método simple de selección estocástico en el cual se asigna una parte proporcional y la suma de todos los porcentajes dan como resultado la unidad. Entonces a cada individuo le corresponde una rebanada de la ruleta y la selección de un individuo será proporcional a su valor de aptitud. Por lo tanto, la aptitud de cada individuo es proporcional a su probabilidad de selección. Y por ser una probabilidad se cumple la ecuación 5.1.

$$\sum_{i=1}^{n} p_i = 1 \tag{5.1}$$

A cada individuo se le asigna una probabilidad de ser seleccionado como padre la cual es proporcional al valor de su función objetivo y se encuentra en el intervalo $[0, 1]$. Por lo tanto, para cada individuo i con un valor asociado de aptitud $c(i)$ se le asigna la probabilidad p_i de ser seleccionado, calculada mediante la ecuación 5.2.

$$p_i = \frac{c(i)}{\sum_{i=1}^{n} c(i)} \tag{5.2}$$

Muestreo estocástico universal (*Stochastic Universal Sampling)*

Este método fue desarrollado por Baker en 1987 y basado en el método de la ruleta, el cual utiliza un conjunto de n individuos.

Al igual que en el método anterior, se asigna un tamaño de la rebanada a cada individuo de acuerdo al valor de la probabilidad, el cual es proporcional al valor en la función objetivo. Después se forma una segunda ruleta con m-espacios igualmente separados entre sí, donde *m* es el número de veces que se gira la primera ruleta. Entonces se gira la primera ruleta y se va seleccionando un individuo en cada iteración, marcando en cada posición de la segunda ruleta el individuo seleccionado. Si se seleccionan *t* veces al mismo individuo entonces *t* veces se registra en la segunda ruleta. Este método garantiza que ningún individuo es seleccionado menos de $\lfloor \nu \rfloor$ y no más de $\lceil \nu \rceil$ veces. Donde ν es el tamaño del espacio.

5.3.2. Operadores de cruce o recombinación

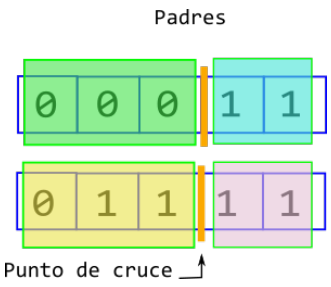
Los operadores de recombinación son importantes para la creación de los descendientes, ya que conforman a las nuevas generaciones. Debido a que es de gran importancia la diversidad de las generaciones se opta por asignar una probabilidad de cruce entre 0.6 y 1.0 [4].

Existen distintas estrategias para realizar el cruce en los cromosomas las cuales se explican a continuación.

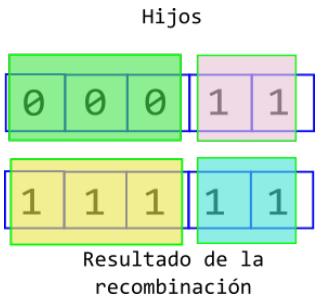
Cruce de un punto

Este tipo de cruce es uno de los más sencillos. Se selecciona de manera aleatoria dos individuos que representan a los padres y genera un corte aleatorio en los cromosomas para producir la recombinación mediante el intercambio de posiciones de los genes. Un ejemplo para esto se muestra en la figura 5.5 donde primero se tienen los dos candidatos padres y se elige el punto de cruce que está señalado en la figura 5.5a.

Mediante la recombinación de los dos cromosomas padres se construyen los dos cromosomas hijos de la figura 5.5b haciendo lo siguiente: el primer hijo está conformado por el gen color verde que se obtuvo de los primeros 3 alelos del primer padre y del segundo padre se obtuvo el gen de color lila que son los últimos 2 alelos. Para el segundo hijo el gen color naranja está conformado de los primeros 3 alelos del segundo padre y el gen color azul está conformado con los dos 2 alelos del primer padre.



(a) Se selecciona de manera aleatoria el punto de corte para ambos cromosomas.

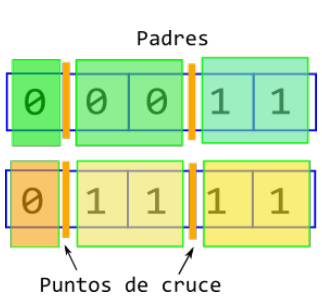


(b) Se genera la recombinación de los genes para crear los nuevos cromosomas de los hijos.

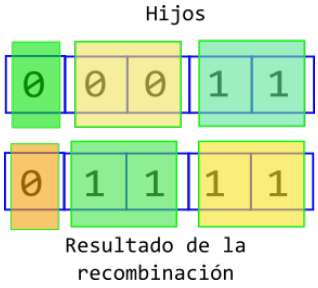
Figura 5.5: Ejemplo de recombinación usando cruce de un punto. Generado en *Inkscape*.

Cruce de dos puntos

Ésta es otra forma de recombinar los individuos en donde se eligen dos puntos de corte aleatorios distintos que no deben coincidir con el extremo de alguno de los cromosomas. Por lo tanto, ahora se tienen tres segmentos por cada cromosoma los cuales se recombinan para la creación de los dos hijos. Esto se muestra en la figura 5.6.



(a) Se seleccionan de manera aleatoria los puntos de corte para ambos cromosomas.



(b) Se genera la recombinación correspondiente para crear los nuevos cromosomas de los hijos.

Figura 5.6: Ejemplo de recombinación usando cruce de dos puntos. Generado en *Inkscape*.

Cruce uniforme

Este tipo de cruce es distinto a los dos anteriores, cada gen que conforma al individuo tiene las mismas probabilidades de pertenecer a uno u otro padre. Se utiliza un vector cromosoma auxiliar donde se le asigna un valor de 0 o 1 a cada gen del cromosoma. Si se encuentra el valor de 1 en la misma entrada del cromosoma del primer padre se copia ese valor, en otro caso cuando hay un valor de 0 se toma el valor de la entrada correspondiente del segundo padre. Para la construcción del segundo hijo se sigue el razonamiento contrario.

5.3.3. Operadores de mutación

Anteriormente se mencionó que este operador tiene como fin provocar una alteración o variación en el individuo con probabilidades menores o iguales a 0.1 incluso hay artículos donde recomiendan valores en el rango de 0.0001 y 0.005 [23]. La mutación sirve para tener diversidad entre los individuos y no se tenga un valor prematuro de aptitud convergente en la población.

Hay dos mutaciones conocidas y comúnmente usadas:

- Reemplazo aleatorio: de manera aleatoria se realiza una variación a un gen del cromosoma.
- Intercambio de valores para algunos alelos del cromosoma.

5.3.4. Criterios de reemplazo

A lo largo del proceso de la heurística se generan nuevos individuos mediante la recombinación entre cromosomas entonces se debe contar con algún criterio que permita saber en que momento un nuevo individuo puede sustituir alguno de la población actual. Para esto existen distintas maneras de realizar el reemplazo:

- Aleatorio: seleccionar de manera aleatoria el individuo de la población actual que será sustituido por el individuo creado.
- Reemplazando a los padres: se integran los descendientes creados en la siguiente generación.
- Reemplazo de similares: si al comparar los valores de aptitud hay similares entonces se reemplazan con cierta probabilidad la cantidad necesaria de individuos.
- Reemplazo de peores individuos: se elige un subconjunto de individuos con un valor de aptitud malo y se seleccionan aleatoriamente los que serán sustituidos por los nuevos descendientes.

5.3.5. Operadores de evaluación

La funcionalidad de estos operadores es conocer qué tan bien o mal están calificados los individuos de la población actual. En general, una función de evaluación establece una medida numérica de la bondad de ajuste de cierta solución. Esta función se encarga de asignarle una probabilidad a cada individuo la cual representa la probabilidad de supervivencia hasta la reproducción y se pondera con el número total de descendientes en la actual generación. Se tienen cuatro maneras de calcular la función de aptitud [20]:

Función de aptitud pura

r(i,t): esta función es una medida de ajuste establecida de acuerdo a la terminología de cada problema. La ecuación 5.3 se aplica para cada individuo de la población, donde se establece el cálculo del valor de bondad del individuo *i* en la generación *t*.

$$r(i,t) = \sum_{j=1}^{N_c} |s(i,j) - c(i,j)| \tag{5.3}$$

Donde:

$s(i,j)$: valor esperado para el individuo i en el caso j
$c(i,j)$: valor obtenido para el individuo i para el caso j
N_c : número de casos

Para los problemas de maximización los individuos que obtengan un valor grande serán los mejor calificados. Y en el caso de los problemas de minimización, los individuos que nos interesan son los que obtengan un valor de aptitud pequeño.

Función de aptitud estandarizada

s(i,t): esta función de evaluación se define en la ecuación 5.4, donde se utiliza la función de aptitud pura para un problema de minimización, mientras que para un problema de maximización se resta una cota superior r_{max} del valor de la función de aptitud pura.

$$s(i,t) = \begin{cases} r(i,t) & \text{minimización} \\ r_{max} - r(i,t) & \text{maximización} \end{cases} \tag{5.4}$$

Un individuo i será mejor calificado si para cualquier otro individuo j se cumple que $s(i,t) < s(j,t)$.

Función de aptitud ajustada

a(i,t): esta función se define en la ecuación 5.5, utilizando una transformación de la función de aptitud estandarizada.

$$a(i,t) = \frac{1}{1+s(i,t)} \tag{5.5}$$

Observando que los valores se encuentra en el intervalo $(0,1]$. Los individuos con un valor de aptitud ajustada cercano o igual a uno son los mejor calificados.

Función de aptitud normalizada

n(i, t): esta función de evaluación se define en la ecuación 5.6 donde n es el número total de individuos de la población. Se compara el valor de la función de aptitud ajustada del individuo con los demás valores de las soluciones.

$$n(i,t) = \frac{a(i,t)}{\sum_{k=1}^{n} a(k,t)} \tag{5.6}$$

Estos valores se encuentran en el intervalo $(0,1]$ y los mejores individuos tienen un valor cercano a 1. En comparación de las funciones de aptitud anteriores, esta función de evaluación indica cuál o cuáles de las soluciones destacan más de entre toda la población debido a la normalización. Y la suma de los valores $n(i,t)$ de todos los individuos son igual a 1.

5.4. Algoritmo genético

1. Comenzar con una población de tamaño n generada aleatoriamente.
2. Calcular la aptitud de cada individuo de la población.
3. Repetir lo siguiente hasta crear los n descendientes:
    i) Seleccionar un par de individuos de la población que serán los padres los cuales serán elegidos de acuerdo al valor de aptitud asignado. Un individuo puede ser seleccionado

más de una vez para reproducirse.

ii) Elegir el operador de cruce. Se genera un número aleatorio uniforme $\mathrm{p_c}$ que designa el o los puntos de cruce para construir los dos descendientes.

iii) Aplicar el operador de mutación a los descendientes con cierta probabilidad $\mathrm{p_m}$ y después reacomoda los cromosomas resultantes en la nueva población.

4. Utilizar el operador de reemplazo en la población actual.

5. Ir al paso 2.

Entonces en cada iteración del proceso se estará obteniendo una nueva población que serán las nuevas generaciones. Para los algoritmos genéticos se recomienda generar entre 50 y 500 o más generaciones. Al conjunto completo de generaciones se le llama *corrida*. Es importante recalcar que como el proceso es aleatorio, en cada corrida o ejecución se obtendrán distintos valores de aptitud.

5.5. Algoritmo genético aplicado al PAV

Las soluciones del PAV están definidas por los ciclos hamiltonianos de una gráfica completa. Se explican las analogías correspondientes:

- Alelo: es cada ciudad que forma parte de la solución.
- Gen: un subconjunto de ciudades o bien la solución parcial.
- Individuo (cromosoma): es la solución total o el ciclo hamiltoniano que se construye.
- Población: el conjunto de soluciones.
- Padres: son los tours seleccionados del conjunto de soluciones candidatas de los cuales pueden crear hijos.
- Generación: es el conjunto de soluciones creadas y seleccionadas que conforman una población.
- Función de aptitud: función que asigna un valor a cada solución, puede ser la función objetivo.

- Mutación: posible variación o alteración aleatoria aplicada al tour, por ejemplo el intercambio aleatorio de la posición de dos ciudades.
- Elitismo: selección de las soluciones con mejor valor de aptitud que son candidatas para generar la próxima generación.

Los siguientes pasos permiten adaptar el algoritmo genético aplicado al PAV:

1. Crear una población inicial de tamaño n.
2. Calcular el valor de aptitud para cada individuo.
3. Seleccionar un subconjunto de soluciones candidatas para formar parte del conjunto de apareamiento.
4. Realizar la recombinación entre los cromosomas padres.
5. Aplicar mutación.
6. Repetir hasta generar el número de generaciones establecidas.

5.5.1. Construcción del algoritmo genético aplicado al PAV

Creación de poblaciones. Para la creación de la primera generación o población inicial se puede generar de manera aleatoria, creando el número de tours necesarios.

Calcular el valor de aptitud para la población. El cálculo de la aptitud se define según el contexto del problema, en este caso se asigna la función de aptitud normalizada.

Para la instancia 3.1a que se abordó en el presente trabajo con cinco ciudades, se construye una población de diez soluciones generadas de manera aleatoria.

	Rutas generadas	Distancia	Valor de aptitud
1	[4,1,2,3,5]	555	0.1709
2	[5,2,3,4,1]	540	0.1865
3	[4,5,1,3,2]	720	0.0000
4	[4,5,2,3,1]	640	0.0829
5	[3,1,5,2,4]	670	0.0518
6	[4,2,3,5,1]	605	0.1191
7	[4,2,5,3,1]	645	0.0777
8	[2,5,4,1,3]	690	0.0310
9	[4,2,1,5,3]	625	0.0984
10	[5,3,2,4,1]	545	0.1813

Tabla 5.1: Generación de población de tamaño n=10

En la tabla 5.1 se colocan las soluciones obtenidas junto con la distancia total y su valor de aptitud. Y se observa que el mejor valor de aptitud es el de la solución 2 con una distancia de 540 y un valor de aptitud de 0.1865, estos valores indican que hay una alta probabilidad de ser solución candidata para el grupo de apareamiento, mientras que la solución 3 tiene un valor de aptitud de 0 de manera que tiene una probabilidad baja para reproducirse.

Selección del grupo de apareamiento. Se elige la selección por ruleta simple. Entonces si el valor de probabilidad asignado es grande, casi cercano a uno, significa que esa solución tiene un mejor valor de aptitud y por lo tanto es mejor calificado para reproducirse. Se seleccionan las k soluciones padres.

En la tabla 5.1 se seleccionan 4 soluciones con mejor valor de aptitud y mediante la selección por ruleta simple con reemplazo se establece el conjunto de apareamiento.

	Padres	Distancia	Valor de aptitud
1	[5,2,3,4,1]	540	0.1865
2	[5,2,3,4,1]	540	0.1865
3	[4,2,3,5,1]	605	0.1191
4	[4,1,2,3,5]	555	0.1709

Tabla 5.2: Elección de 4 rutas-padres.

En la tabla 5.2 se observa que las primeras dos soluciones padres son iguales, es decir, se eligió la misma solución dos veces ya que el valor de aptitud es alto.

Se realiza el cruce de un punto con una probabilidad de 0.9 y se tiene como resultado la población que se muestra en la tabla 5.3.

	Rutas	Distancia
1	[4,2,3,5,1]	605
2	[5,2,3,4,1]	540
3	[1,2,3,4,5]	570
4	[5,2,3,4,1]	540
5	[5,2,3,4,1]	540
6	[5,2,3,4,1]	540
7	[1,2,3,4,5]	570
8	[5,2,3,4,1]	540
9	[4,2,3,1,5]	640
10	[5,2,3,4,1]	540

Tabla 5.3: Nueva población obtenida después de la recombinación de las rutas-padres.

Aplicación de la mutación

Para la mutación se elige utilizar el intercambio aleatorio de las posiciones de dos ciudades como se muestra en la figura 5.7. Se tiene la solución de la izquierda a la cual se le aplica el intercambio aleatorio de posiciones de las ciudades 1 y 3 entonces, la ciudad 1 se mueve a la posición de la ciudad 3 y viceversa, resultando la nueva solución de la derecha.

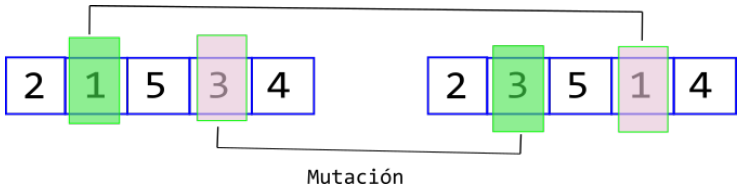


Figura 5.7: Se seleccionan dos ciudades para intercambiarse de lugar. Generado en *Inkscape*.

Asignando el valor al parámetro de probabilidad de mutación recomendado de 0.10, se obtiene la población final y se muestra en la tabla 5.4.

	Rutas	Distancia
1	[5,2,3,4,1]	540
2	[2,5,3,4,1]	540
3	[5,2,3,4,1]	540
4	[5,3,2,4,1]	540
5	[4,2,5,3,1]	645
6	[5,2,3,4,1]	540
7	[4,2,3,1,5]	640
8	[4,1,2,3,5]	555
9	[4,1,2,3,5]	555
10	[5,2,3,4,1]	540

Tabla 5.4: Población de 10 rutas-individuos después de aplicar mutación.

Se observa que en la tabla 5.4 hay varias soluciones cuya distancia es de 540 unidades por lo que se intuye que la población converge a una cierta distancia en pocas iteraciones y esto es debido a que el tamaño de la instancia no es muy grande. El valor al que converge corresponde a la distancia óptima.

Capítulo 6

Resultados computacionales

En este capítulo se realiza la comparación computacional entre las distintas implementaciones utilizando un conjunto de instancias tomadas del repositorio [21].

TSPLIB [21] es una librería de datos que brinda varias instancias para el problema del agente viajero (Traveling Salesman Problem). El nombre de las bases están constituidas en dos partes: en la primera hay letras y en la segunda parte hay valores numéricos que indican el número de ciudades que las constituyen. Las instancias contienen una lista de números flotantes que representan las coordenadas (x, y) de los diferentes puntos de las n ciudades.

6.1. Obtención de instancias

Las instancias a considerar tienen entre 16 y 200 ciudades las cuales se descargaron de la librería TSPLIB [21]. Además, todas pertenecen al PAV simétrico y los mejores valores conocidos de las distancias están reportados en distintos artículos, páginas y medios electrónicos así como en [21]. Las instancias elegidas son las siguientes: att48, bayg29, berlin52, ch150, eil101, KroA100, kroA150, kroA200, kroB100, kroB200, kroC100, kroD100, kroE100, lin105, rat195, ulysses16 y ulysses22. Estas diferentes instancias contienen coordenadas de distintos lugares, por ejemplo, la instancia ulysses16 tiene 16 ciudades que corresponden a la Odisea de Ulises; bayg29 son las 29 ciudades de Baviera, Alemania; att48 representa 49 capitales de EE.UU; berlin52 son 52 ubicaciones de la ciudad de Berlin.

Para trabajar con las coordenadas de las instancias se calcularon las distancias de entre todo par de ciudades y recordando que al ser la gráfica simétrica la distancia entre las ciudades i y j es la misma que la distancia de j a i. La distancia utilizada es la norma euclidiana que se muestra en la ecuación 6.1:

$$d_{ij} = \sqrt{(x_i - x_j)^2 - (y_i - y_j)^2} \tag{6.1}$$

6.2. Análisis comparativo de las implementaciones

La implementación se realizó en el lenguaje de programación Phyton y se ejecutaron en Windows 10 con un CPU core i5 y 8 GB de RAM. Para realizar la comparación de los resultados obtenidos se calculó la siguiente tasa de error porcentual 6.2 :

$$\text{Error} = \frac{\text{Mejor valor obtenido} - \text{Óptimo}}{\text{Óptimo}} \text{x}100 \tag{6.2}$$

Donde el valor *Óptimo* es el mejor conocido y reportado, en este trabajo se tomaron los valores de los siguientes trabajos [1] y [2].

6.2.1. Resultados aplicando vecino más cercano

Para la prueba computacional de esta heurística se elige para todas las instancias la ciudad inicial 1 con el fin de reportar el resultado de la distancia del ciclo hamiltoniano. La tabla 6.1 contiene los resultados para las distintas instancias, la segunda columna indica el número de ciudades, la tercera columna tiene la distancia total del ciclo hamiltoniano y la cuarta el tiempo empleado en ejecutar la función para ilustrar el resultado se muestran en la figura 6.1 cuatro ciclos hamiltonianos resultantes de las instancias: Ulysses16, Att48, berlin52, ch150 y kroB100.

Instancia	No. Ciudades	Distancia	Tiempo
Att48	48	40526.42	0.0000
Bayg29	29	10211.18	0.0037
berlin52	52	8980.92	0.0010
ch150	150	8194.61	0.0161
eil101	101	825.24	0.0061
KroA100	100	26856.39	0.0060
kroA150	150	33609.87	0.0020
kroA200	200	35798.41	0.0341
kroB100	100	29155.04	0.0110
kroB200	200	36981.59	0.0382
kroC100	100	26327.36	0.0128
kroD100	100	26950.46	0.0111
kroE100	100	27587.19	0.0059
lin105	105	20362.76	0.0189
rat195	195	2761.96	0.0294
Ulysses16	16	104.73	0.0000
Ulysses22	22	89.64	0.0098

Tabla 6.1: Resultados obtenidos para vecino más cercano.

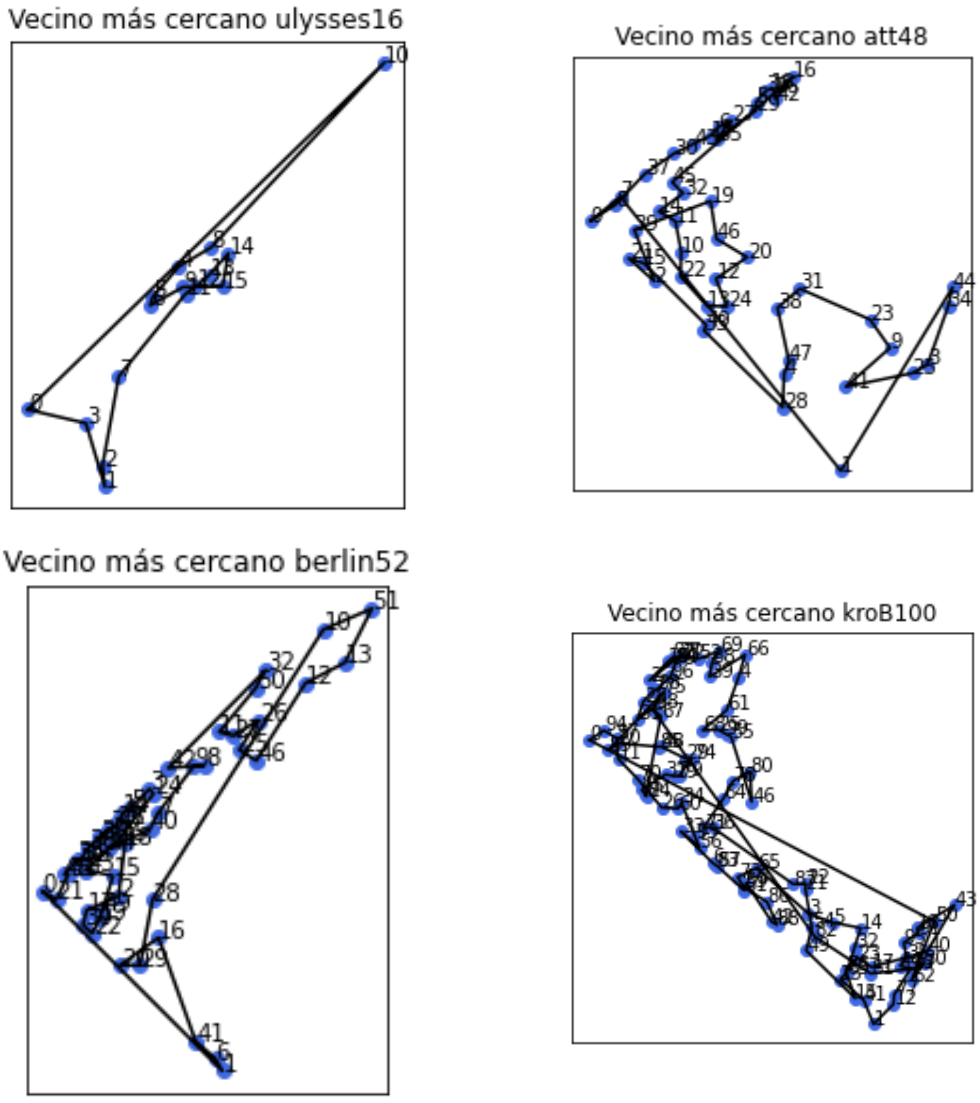


Figura 6.1: Ciclos resultantes de iniciar en la ciudad 1 para las bases ulysses16, att48, berlin52 y kroB100.

6.2.2. Resultados búsqueda local con 2-opt

Con el fin de obtener una solución con menor distancia se realizaron dos búsquedas locales utilizando la estrategia del 2-intercambio. La búsqueda local BL (1) realiza el 2-intercambio hasta que encuentre una mejor solución a la inicial, mientras que la búsqueda local BL (2) dada una solución le aplica el 2-intercambio y con la mejor solución que encuentra vuelve a realizar el

2-intercambio hasta cumplir el número de iteraciones.

En la tabla 6.2 se utiliza la instancia berlin52 para comparar los resultados que se obtienen utilizando distintas ciudades iniciales, entonces puede verse que en la primera columna muestra la ciudad inicial seleccionada para el tour, en la segunda columna esta la distancia total del tour encontrado utilizando vecino cercano, en la tercera columna están los resultados obtenidos al aplicar la BL (1) y en la cuarta columna los resultados correspondientes de aplicar BL (2).

Ciudad inicial	Distancia del tour con VMC	Distancia del ciclo con BL (1)	Distancia del ciclo con BL (2)
1	8980.92	8384.19	8615.51
3	9709.42	8880.38	9071.00
24	9099.35	8327.42	8928.12
50	9252.40	8339.64	8728.71

Tabla 6.2: Comparación de distancias obtenidas para vecino más cercano y búsqueda local iniciando en distintas ciudades.

De las dos búsquedas locales se elige BL (1) para compararse con los resultados obtenidos de la heurística de vecino más cercano, entonces en la tabla 6.3 se tienen los resultados para todas las instancias. Se tienen los siguientes datos: en la primera columna el nombre de la instancia, en la segunda columna está el mejor resultado conocido, en la tercera la distancia del tour encontrado usando vecino más cercano y la cuarta columna la distancia usando la búsqueda local elegida, para todas las instancias se elige la ciudad inicial 1.

Se observa que para algunas de las instancias como: att48, Bayg29, kroA100, kroA150, kroA200, kroE100 y ulysses16 la distancia del tour obtenido por vecino más cercano disminuye notoriamente después de aplicar búsqueda local con esa solución obtenida. Por ejemplo en la tabla 6.3, para la instancia att48 la distancia aplicando vecino más cercano es de 40526.42 y la distancia obtenida aplicando búsqueda local es de 37196.82 entonces se obtiene una disminución de 3329.60 de manera que el nuevo tour encontrado tiene menor distancia que el tour inicial.

Instancia	Mejor valor conocido	Distancia con VMC	Distancia con BL
att48	33523.70	40526.42	37196.82
Bayg29	9074.14	10211.18	9791.77
berlin52	7542.00	8980.92	8728.71
ch150	6528.00	8194.61	8060.03
eil101	629.00	825.24	731.48
kroA100	21282.00	26856.39	24192.28
kroA150	26524.00	33609.87	31739.42
kroA200	29368.00	35798.41	33288.31
kroB100	22141.00	29155.04	28594.19
kroB200	29437.00	36981.59	35587.08
kroC100	20749.00	26327.36	25614.99
kroD100	21294.00	26950.46	26460.21
kroE100	22068.00	27587.19	25981.13
lin105	14379.00	20362.76	19590.68
rat195	2323.00	2761.96	2701.97
Ulysses16	74.10	104.73	83.61
Ulysses22	75.66	89.64	85.64

Tabla 6.3: Comparación de distancias entre la solución del vecino más cercano (VMC) y búsqueda local (BL).

Y para algunas es evidente que la elección de la ciudad inicial es esencial para la solución. Entonces ahora se muestra en la tabla 6.4 la prueba computacional donde se eligen las ciudades iniciales de manera aleatoria y se muestran las columnas de las distancias asociadas.

Instancia	Mejor valor conocido	Ciudad inicial del tour				Distancias			
att48	33523.70	13	25	40	33	35000.87	37262.05	35900.79	35243.01
Bayg29	9074.14	7	4	29	15	9848.71	9994.02	9913.33	9707.42
berlin52	7542.00	38	16	18	36	8279.42	8573.23	7678.90	9012.01
ch150	6528.00	14	50	104	21	7484.20	7555.28	7472.13	7181.56
eil150	629.00	75	1	34	56	719.70	689.39	734.43	712.77
kroA100	21282.00	74	1	34	49	23589.52	22601.38	24915.40	23624.81
kroA150	26524.00	14	50	104	21	29960.35	29549.94	30561.75	30540.30
kroA200	29368.00	137	52	29	109	34347.16	34573.87	33419.97	31032.52
kroB100	22141.00	74	1	34	49	26072.74	24600.34	24129.70	23818.35
kroB200	29437.00	137	52	29	109	33896.79	33629.54	33807.15	32500.62
kroC100	20749.00	74	1	34	49	22684.93	23632.79	23038.24	22118.06
kroD100	21294.00	74	1	34	49	23703.24	22830.83	23787.97	23356.92
kroE100	22068.00	72	34	32	7	23969.08	24403.11	26445.84	25540.90
lin105	14379.00	97	90	70	92	15042.46	16185.69	16060.98	16393.85
rat195	2323.00	177	121	174	192	2580.89	2501.60	2657.78	2619.96
Ulysses16	74.10	14	4	8	10	74.83	84.79	80.26	74.97
Ulysses22	75.66	21	6	17	18	76.86	80.06	84.52	76.78

Tabla 6.4: Comparación de resultados usando búsqueda local con distintos inicios en el tour.

En la siguiente tabla 6.5 se muestran las tasas de error para cada una de las soluciones.

Instancia	Mejor valor conocido	Ciudad inicial del tour				Distancias			
att48	33523.70	13	25	40	33	4.41	11.15	7.09	5.13
Bayg29	9074.14	7	4	29	15	8.54	10.14	9.25	6.98
berlin52	7542.00	38	16	18	36	9.78	13.67	1.82	19.49
ch150	6528.00	14	50	104	21	14.65	15.74	14.46	10.01
eil101	629.00	75	1	34	56	14.42	9.60	16.76	13.32
kroA100	21282.00	74	1	34	49	10.84	6.20	17.07	11.01
kroA150	26524.00	14	50	104	21	12.96	11.41	15.22	15.14
kroA200	29368.00	137	52	29	109	16.95	17.73	13.87	5.67
kroB100	22141.00	74	1	34	49	17.76	11.11	8.98	7.58
kroB200	29437.00	137	52	29	109	15.15	14.24	14.85	10.41
kroC100	20749.00	74	1	34	49	13.90	11.03	11.03	6.60
kroD100	21294.00	74	1	34	49	11.31	7.22	11.71	9.69
kroE100	22068.00	72	34	32	7	8.61	10.58	19.84	15.74
lin105	14379.00	97	90	70	92	4.61	12.56	11.70	14.01
rat195	2323.00	177	121	174	192	11.10	7.69	14.41	12.78
Ulysses16	74.10	14	4	8	10	0.99	14.43	8.31	1.17
Ulysses22	75.66	21	6	17	18	1.59	5.82	11.71	1.48

Tabla 6.5: Comparación de tasas de error para los resultados de la tabla anterior.

6.2.3. Resultados aplicando recocido simulado

Para generar las soluciones vecinas se uso la idea del intercambio de dos ciudades.

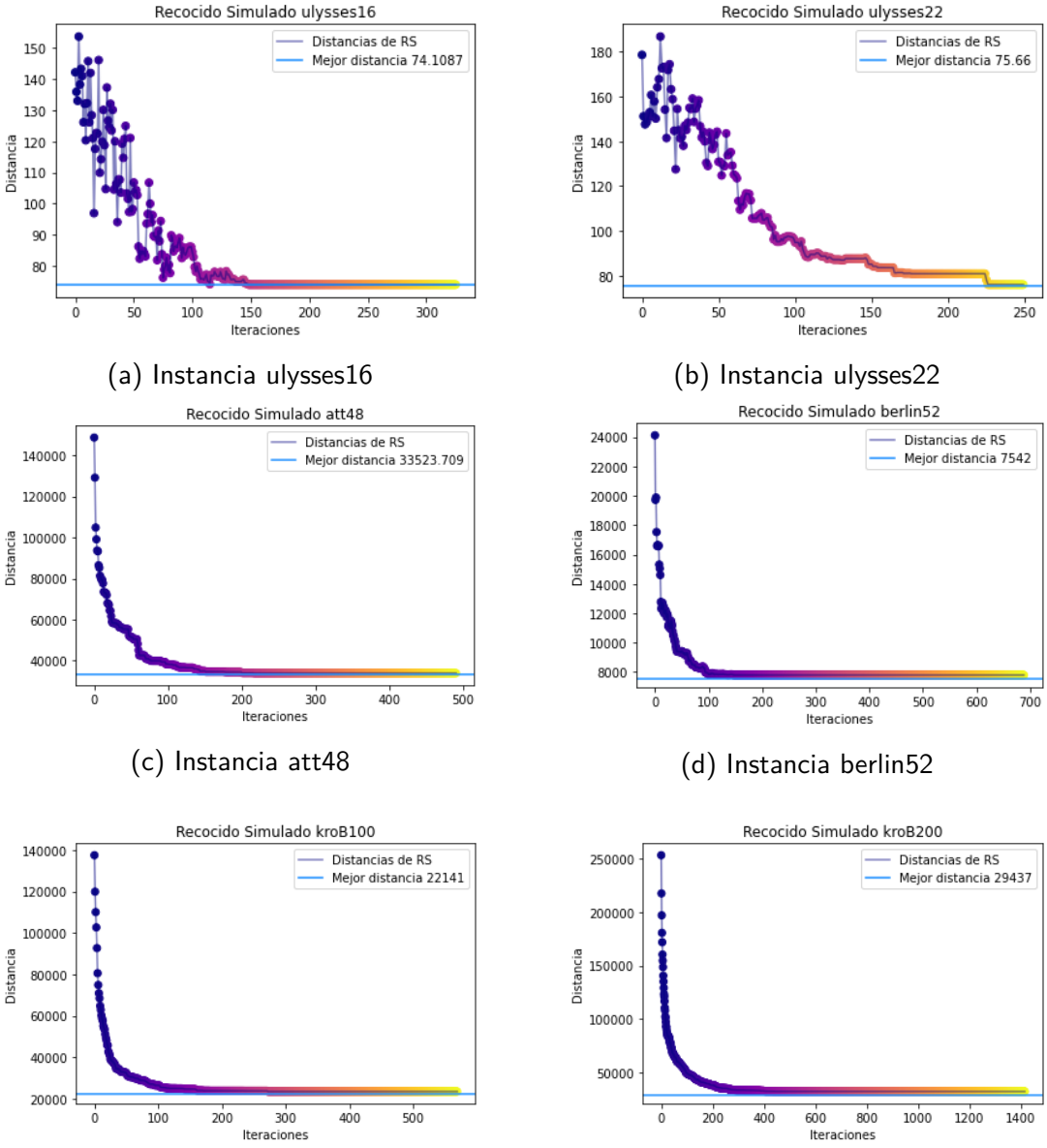
Los siguientes hiperparámetros de la heurística fueron definidos después de hacer varias pruebas con cada instancia y se quedaron los valores que daban un menor porcentaje de error, dando como resultado los siguientes rangos para todas las instancias. Las temperaturas iniciales T_0 elegidas se encuentran en un rango de 20 a 250. Los valores de las tasas de enfriamiento α están en un rango de 0.97 a 0.99. Y el número de repeticiones $nrep$ fueron de 5 a 200. El valor elegido dependió del número de ciudades de la instancia, ya que para un número pequeño de ciudades el número de iteraciones fueron menos que para una instancia con más ciudades.

Instancia	Distancia(metros) (metros)		Tiempo	Error
	Valor óptimo conocido	Mejor valor obtenido	(seg)	(%)
att48	33523.70	33955.83	0.05	1.29
Bayg29	9074.14	9127.23	0.15	0.59
berlin52	7542.00	7799.48	0.10	3.41
ch150	6528.00	6809.51	0.43	4.31
eil101	629.00	676.35	0.54	7.53
KroA100	21282.00	22750.41	0.10	6.90
kroA150	26524.00	28526.04	0.51	7.55
kroA200	29368.00	31686.02	0.49	7.89
kroB100	22141.00	23447.23	0.16	5.90
kroB200	29437.00	32347.35	0.65	9.89
kroC100	20749.00	22123.13	0.11	6.62
kroD100	21294.00	22843.15	0.17	7.28
kroE100	22068.00	23621.68	0.27	7.04
lin105	14379.00	15610.75	0.16	8.57
rat195	2323.00	2545.76	0.85	9.59
Ulysses16	74.10	74.11	0.02	0.00
Ulysses22	75.66	76.02	0.00	0.48

Tabla 6.6: Resultados del recocido simulado aplicado a las instancias de la librería TSPLIB [21].

La tabla 6.6 muestra los resultados obtenidos de las instancias, en la segunda columna se escriben los mejores valores de distancia conocidos, la tercera columna tiene los valores obtenidos por la implementación, la cuarta columna contiene el tiempo empleado en obtener el resultado y la quinta columna contiene el error porcentual calculado como se indica en la ecuación 6.2.

Se observa que los errores porcentuales son poco significativos para las instancias con un menor número de ciudades como se aprecia para las instancias ulysses16, ulysses22, att48, uayg29 y berlin52 aunque el error de ésta ya es un poco más notorio en comparación de los anteriores. Mientras que para las instancias con más de 100 ciudades los errores porcentuales son más relevantes ya que no se acercan lo suficiente a los mejores valores conocidos. A continuación se muestran las imágenes de la evolución de algunos resultados a través de las iteraciones en donde la temperatura decrece y el valor de las distancias de los ciclos hamiltonianos encontrados también disminuyen.



(a) Instancia ulysses16

(b) Instancia ulysses22

(c) Instancia att48

(d) Instancia berlin52

(e) Instancia kroB100

(f) Instancia kroB200

Figura 6.2: Recocido Simulado vs. Mejor valor conocido

6.2.4. Resultados aplicando Algoritmos Genéticos

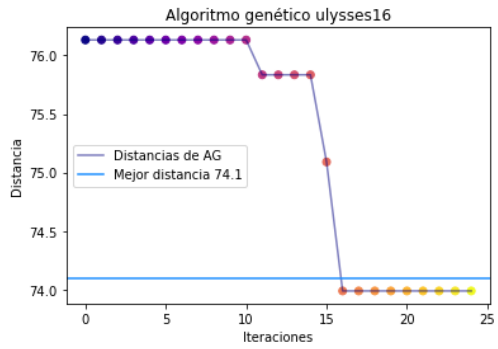
Los resultados de las pruebas computacionales que se presentan en la tabla 6.7 se obtuvieron después de distintas calibraciones en los parámetros. El tamaño de la población elegido para las distintas instancias se encuentra en un rango de entre 20 y 110, la probabilidad de cruce fue del

0.9, la probabilidad de mutación tiene el valor del 0.1 y el número de iteraciones dependió del tamaño de la instancia ya que el espacio de soluciones es más grande y se observó que al realizar más iteraciones el resultado mejoraba para estas instancias, y se acotan en un rango de 25 a 450 veces.

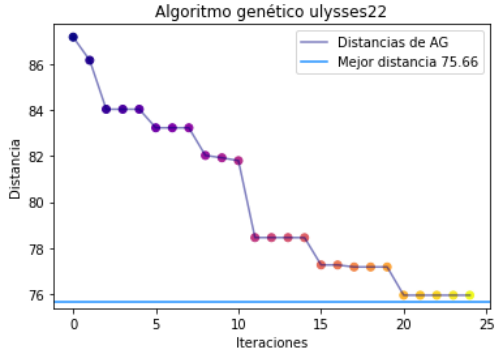
Instancia	Distancia (metros)		Tiempo	Error
	Valor óptimo conocido	Mejor valor obtenido	(seg)	(%)
att48	33523.70	35175.54	48.22	4.93
Bayg29	9074.14	9094.64	1.50	0.23
Berlin52	7542.00	7968.00	23.91	5.64
ch150	6528.00	6892.40	192.64	5.58
eil101	629.00	672.14	1246.32	6.85
KroA100	21282.00	21491.94	403.69	0.99
kroA150	26524.00	29112.31	1764.70	9.76
kroA200	29368.00	33175.67	2447.66	12.97
kroB100	22141.00	24185.08	584.78	9.23
kroB200	29437.00	32299.28	2200.28	9.72
kroC100	20749.00	22404.73	217.45	7.98
kroD100	21294.00	22983.38	1360.56	7.93
kroE100	22068.00	23560.21	218.53	6.76
lin105	14379.00	14962.53	223.36	4.06
rat195	2323.00	2528.12	399.10	8.83
Ulysses16	74.10	74.00	0.17	-0.13
Ulysses22	75.66	75.95	0.77	0.39

Tabla 6.7: Resultados de algoritmos genéticos aplicado a las instancias de la librería TSPLIB [21]

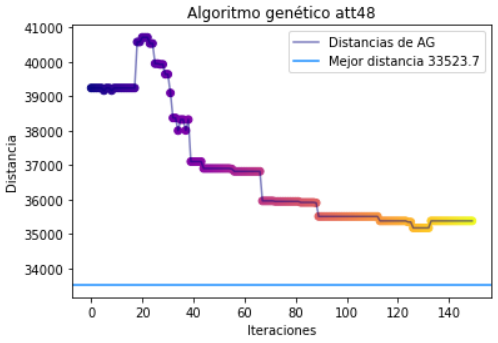
En la tabla 6.7 se observa en la quinta columna que los errores porcentuales para las instancias ulysses16, ulysses22 y bayg29 son lo más pequeños con una tasa del -0.13, 0.39 y 0.23 respectivamente, mientras que para las instancias kroA200 y kroA150 sus tasas de error son las más grandes con un valor de 12.97 y 9.76. Notando que los errores porcentuales son más grandes. En la figura 6.3 se muestra que a través de las iteraciones la distancia que se encuentra es menor a la inicial.



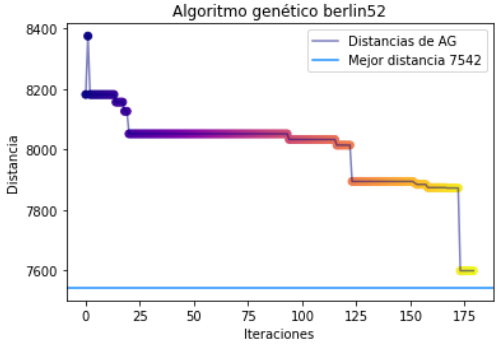
(a) Instancia ulysses16



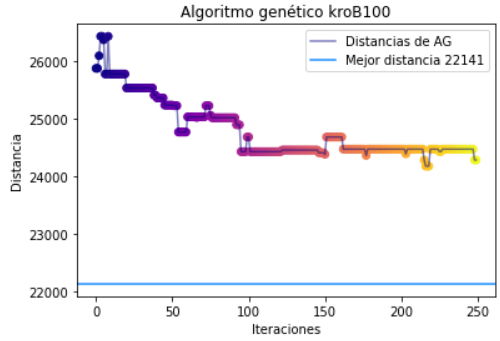
(b) Instancia ulysses22



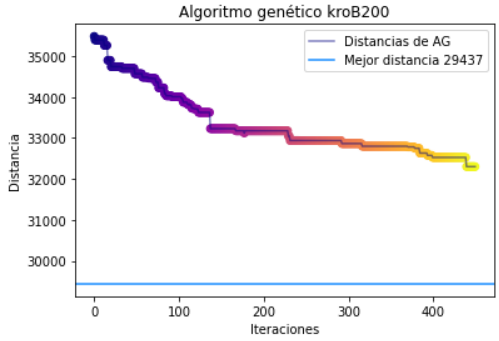
(c) Instancia att48



(d) Instancia berlin52



(e) Instancia kroB100



(f) Instancia kroB200

Figura 6.3: Algoritmos genéticos vs. Mejor valor conocido

6.3. Resumen de los resultados

En las tablas 6.8 y 6.9 se muestran las comparaciones de los errores porcentuales obtenidos y los tiempos empleados para las implementaciones de vecino más cercano, búsqueda local, recocido simulado y algoritmos genéticos.

Instancia	Vecino más cercano	Búsqueda Local	Recocido Simulado	Algoritmos Genéticos
att48	20.89	10.96	1.29	4.93
Bayg29	12.53	7.91	0.59	0.23
berlin52	19.08	15.73	3.41	5.64
ch150	25.53	23.47	4.31	5.58
eil150	31.20	16.29	7.53	6.85
kroA100	26.19	13.67	6.90	0.99
kroA150	26.71	19.66	7.55	12.97
kroA200	21.9	13.35	7.89	12.97
kroB100	31.68	29.15	5.90	9.23
kroB200	25.63	20.89	9.89	9.72
kroC100	26.88	23.45	6.62	7.98
kroD100	26.56	24.26	7.28	7.93
kroD100	26.56	16.09	7.28	7.93
kroE100	25.01	17.73	7.04	6.76
lin105	41.61	36.25	8.57	4.06
rat195	18.90	16.31	9.59	8.83
Ulysses16	41.34	3.12	0.00	-0.13
Ulysses22	18.48	0.33	0.48	0.39
Promedio	26.13	11.49	5.43	5.74

Tabla 6.8: Comparación de errores porcentuales entre los algoritmos.

La tabla 6.8 muestra los resultados de las tasas de error calculadas a cada instancia para comparar el desempeño en cada implementación aplicada. Al final se agregan los promedios de las tasas de error obtenidas para las implementaciones, ordenándolos de manera descendente quedan de la siguiente forma: $26.13\,\%$ vecino más cercano, $11.49\,\%$ búsqueda local, $5.74\,\%$ algoritmos genéticos y $5.43\,\%$ recocido simulado

Instancia	Vecino más cercano	Búsqueda Local	Recocido Simulado	Algoritmos Genéticos
att48	0.00	0.24	0.05	48.22
Bayg29	0.00	0.03	0.15	1.50
berlin52	0.00	0.23	0.10	23.91
ch150	0.03	5.41	0.43	192.64
eil150	0.03	1.67	0.54	1246.32
kroA100	0.03	1.55	0.10	403.69
kroA150	0.03	6.45	0.51	1764.70
kroA200	0.03	14.56	0.49	2447.66
kroB100	0.03	1.75	0.49	1400.66
kroB200	0.03	15.09	0.16	584.78
kroC100	0.03	1.58	0.65	2200.28
kroD100	0.03	1.66	0.11	217.45
kroD100	0.03	1.84	0.17	1360.56
kroE100	0.03	1.70	0.27	218.53
lin105	0.03	1.97	0.16	223.36
rat195	0.03	13.86	0.85	399.10
Ulysses16	0.00	0.00	0.02	0.17
Ulysses22	0.00	0.01	0.00	0.77
Promedio	0.02	3.86	0.29	707.45

Tabla 6.9: Comparación de tiempo empleado.

En la tabla 6.9 se muestra el tiempo empleado de cada instancia usando las diferentes implementaciones, también se agrega el promedio de tiempo empleado para cada una por lo que si se acomodan en orden descendente quedan: algoritmos genéticos, búsqueda local, recocido simulado y vecino más cercano.

Es importante mencionar que utilizar un menor tiempo no implica tener un mejor resultado, ya que en el caso de vecino más cercano el promedio de tiempo utilizado es menor pero la tasa de error no lo es.

Con todos los cálculos anteriores, se concluyen los siguientes puntos:

- La implementación con el menor error porcentual promedio es recocido simulado, con un valor de $5.43\,\%$.
- La implementación con el menor promedio de tiempo utilizado es vecino más cercano, con

un valor de 0.02 segundos.

- La comparación de eficiencia del tiempo entre las heurísticas: recocido simulado y algoritmos genéticos, es significativa ya que los resultados son de 0.29 y 707.45 segundos respectivamente.
- La comparación de los errores porcentuales entre las heurísticas: recocido simulado y algoritmos genéticos, es de $5.43\,\%$ y $5.74\,\%$ por lo que distan de un $.31\,\%$ lo cual es poco significativo al utilizarlas.
- En general para instancias con un mayor número de ciudades el tiempo en obtener un buen valor puede ser muy lento para los algoritmos genéticos, pero no para recocido simulado, esto debido a la forma en cómo generan el espacio de soluciones y los movimientos.
- Recocido simulado tiene un buen desempeño para cualquier instancia, ya que presenta la menor tasa de error promedio del $5.43\,\%$ en un tiempo promedio de 0.29 segundos.

Enlace a GitHub.

Capítulo 7

Conclusiones

Si bien, un problema de optimización combinatoria puede verse fácil de resolver cuando el número de variables, restricciones e iteraciones son pocas, la complejidad se encarga de medir lo difícil que es tratar de resolverlo cuando todo esto aumenta (concluyéndose que los algoritmos determinísticos no son una opción para tratar de encontrar el óptimo en instancias muy grandes).

Se logró comparar la calidad de los costos de las mejores soluciones encontradas para el problema del agente viajero usando las diferentes implementaciones y se observó que las heurísticas son estrategias de gran rendimiento que proporcionan soluciones factibles y con un buen valor para el problema. Asimismo, se comprende la importancia de las heurísticas para la aproximación del óptimo de un problema difícil de resolver, como lo es el problema del agente viajero.

Los resultados finales fueron obtenidos después de una serie de ajustes y pruebas que se aplicaron a los parámetros de las implementaciones logrando establecer rangos y valores específicos que se seleccionaron. Concluyendo que para ciertas instancias los resultados obtenidos fueron cercanos a los resultados de las fuentes consultadas que se tomaron como referencia, pero también se obtuvieron resultados bastantes lejanos para instancias con un mayor número de ciudades.

Después de lo expuesto en el presente trabajo se tiene claro que existen diversas maneras de abordar un problema y que se pudieron haber implementado más técnicas para comparar resultados en cuanto a la eficiencia computacional. Entonces como trabajo a futuro se tiene la intención de mejorar la construcción de las vecindades, ya que es una parte importante.

Para la heurística de algoritmos genéticos se tienen muchas maneras de interactuar con las poblaciones por lo que sería interesante probar más operadores y combinarlos de tal forma que

mejoren el tiempo de ejecución al que se obtuvo. A través del análisis de resultados, se destaca la heurística de recocido simulado, ya que la calidad de sus resultados permitieron alcanzar el mejor rendimiento en tiempo promedio.

Las ideas expuestas anteriormente muestran que el objetivo de la tesis se cumplió mediante las comparaciones que se realizaron con las implementaciones, y también se logró comprender las distintas definiciones necesarias para abordar los problemas que son difíciles de resolver.

Finalmente, se concluye que las heurísticas son estrategias eficientes para resolver problemas difíciles y se debe tener ingenio para realizar las implementaciones de acuerdo al problema con el que se está trabajando.

Referencias

[1] Acosta, N. J. M. Recocido simulado y el algoritmo de selección clonal aplicados al problema del agente viajero. *Res. Comput. Sci. 149*, 8 (2020), 1211–1226.

[2] Ali, Z. A. Concentric tabu search algorithm for solving traveling salesman problem. Master's thesis, Eastern Mediterranean University (EMU)-Doğu Akdeniz Üniversitesi (DAÜ), 2016.

[3] Ayuso, M. d. C. H. *Introducción a la teoría de redes*. Sociedad Matemática Mexicana, 1997.

[4] Beasley, D., Bull, D. R., and Martin, R. R. An overview of genetic algorithms: Part 1, fundamentals. *University computing 15*, 2 (1993), 56–69.

[5] Bektas, T. The multiple traveling salesman problem: an overview of formulations and solution procedures. *Omega 34*, 3 (2006), 209–219.

[6] Burke, E. K., Burke, E. K., Kendall, G., and Kendall, G. *Search methodologies: introductory tutorials in optimization and decision support techniques*. Springer, 2014.

[7] Cook, S. A. The complexity of theorem-proving procedures.

[8] Davendra, D. *Traveling salesman problem: Theory and applications*. BoD–Books on Demand, 2010.

[9] Dowsland, K. A., and Díaz, B. Heuristic design and fundamentals of the simulated annealing. *Revista iberoamericana de inteligencia artificial 7*, 19 (2003), 93–102.

[10] Felipe, Á., Ortuño, M. T., and Tirado, G. The double traveling salesman problem with multiple stacks: A variable neighborhood search approach. *Computers & Operations Research 36*, 11 (2009), 2983–2993.

[11] Gutin, G., and Punnen, A. P. *The traveling salesman problem and its variations*, vol. 12. Springer Science & Business Media, 2006.

[12] Marti, R. Procedimientos metaheurısticos en optimización combinatoria. *Matemátiques, Universidad de Valencia 1*, 1 (2003), 3–62.

[13] Melián, B., Pérez, J. A. M., and Vega, J. M. M. Metaheurísticas: Una visión global. *Inteligencia Artificial. Revista Iberoamericana de Inteligencia Artificial 7*, 19 (2003), 0.

[14] Metropolis, N., Rosenbluth, A. W., Rosenbluth, M. N., Teller, A. H., and Teller, E. Equation of state calculations by fast computing machines. *The journal of chemical physics 21*, 6 (1953), 1087–1092.

[15] Meunier, F., and Sarrabezolles, P. Multicolor traveling salesman problem: approximation and feasibility.

[16] Mezioud, C., and Kholladi, M. K. Research for solving optimization problems of planning and allocation of frequencies for mobile operators. *IJCST 3*, 5 (2012).

[17] Miller, C. E., Tucker, A. W., and Zemlin, R. A. Integer programming formulation of traveling salesman problems. *Journal of the ACM (JACM) 7*, 4 (1960), 326–329.

[18] Mitchell, M. *An introduction to genetic algorithms*. MIT press, 1998.

[19] Papadimitriou, C. H., and Steiglitz, K. *Combinatorial optimization: algorithms and complexity*. Courier Corporation, 1998.

[20] Rabuñal, M. R. G. D., Ramón, J., Dorado, Pazos, J., Alejandro), and Gestal, M. *Introducción a los algoritmos genéticos y la programación genética*. Universidade da Coruña, 2010.

[21] Reinhelt, G. {TSPLIB}: a library of sample instances for the tsp (and related problems) from various sources and of various types. *URL: http://comopt. ifi. uniheidelberg. de/software/TSPLIB95* (2014).

[22] Resende, M. G., and Ribeiro, C. C. *Optimization by GRASP*. Springer, 2016.

[23] Srinivas, M., and Patnaik, L. M. Adaptive probabilities of crossover and mutation in genetic algorithms. *IEEE Transactions on Systems, Man, and Cybernetics 24*, 4 (1994), 656–667.

[24] Talbi, E.-G. *Metaheuristics: from design to implementation*, vol. 74. John Wiley & Sons, 2009.

[25] Zhan, S.-h., Lin, J., Zhang, Z.-j., and Zhong, Y.-w. List-based simulated annealing algorithm for traveling salesman problem. *Computational intelligence and neuroscience 2016* (2016).